jamk

K6-Suorituskykytestaus
CASE: Prestashop

Oskar Sinkkilä

Opinnäytetyö, AMK
Toukokuu 2025
Tieto- ja viestintätekniikan tutkinto-ohjelma

jamk | Jyväskylän ammattikorkeakoulu
University of Applied Sciences

jamk

Kuvailulehti

Sinkkilä Oskar

K6-Suorituskykytestaus CASE: PrestaShop

Jyväskylä: Jyväskylän ammattikorkeakoulu. Toukokuu 2025, 45 sivua.

Tieto- ja viestintätekniikan tutkinto-ohjelma. Opinnäytetyö AMK.

Julkaisun kieli: suomi

Julkaisulupa avoimessa verkossa: kyllä

Tiivistelmä

Opinnäytetyön tarkoituksena oli tutustua suorituskykytestaukseen K6-suorituskykytestaus työkalulla, tutustua PrestaShop verkkokauppa-alustaan. Töitä tehtiin Task Force Tiimissä, joka tuli myös toimimaan tukihenkilöinä Future Factory osallistujille.

Työ toteutettiin CSC cPouta virtuaalikoneilla, joita käytettiin koululla sekä etänä. Töitä tehtiin lähitoteutuksena jouluun asti, ja sen jälkeen suurin osa ajasta töitä tehtiin etänä. Töissä kommunikaatioon käytettiin Discord-keskusteluohjelmaa, ja sinne luotuja palvelimia. Alkututustumisen jälkeen Tesk Force-tiimin jäsenet erikoistuivat eri osa-alueisiin ja asioihin.

Suorituskykytestauksen tulokset saatiin kuormitustestauksella. Huomattiin että PrestaShop-verkkokauppa-alustan suorituskyvyssä olisi parannettavaa. Tieto saatiin tutkimalla kuormitustestien tuloksia, ja tarkemmin sanottuna vasteaikoja, prosenttipisteitä, läpäisyprosentteja sekä aikarajoja.

Johtopäätökset ovat, että PrestaShopin suorituskyvyssä olisi parannettavaa. Mahdollisia parannuksia ovat virtuaalikoneiden tehojen lisääminen ja skaalautuvuuden parantaminen. Kuormitustestien tuloksista näkyi, että suorituskyky kärsii isoilla kävijämäärillä.

Avainsanat (asiasanat)

Testaus, Suorituskykytestaus, Grafana, Prometheus, K6, PrestaShop

Muut tiedot (salassa pidettävät liitteet)

jamk

Description

Sinkkilä, Oskar

K6-performance testing CASE: PrestaShop

Jyväskylä: JAMK University of Applied Sciences, May 2025, 45 pages

Degree Programme in information and communication Technology. Bachelor´s thesis.

Permission for open access publication: yes

Language of publication: Finnish

Abstract

The purpose of this thesis was to explore performance testing using the K6 performance testing tool and to become familiar with the PrestaShop e-commerce platform. The work was carried out as part of the Task Force Team, which also served as support personnel for Future Factory participants.

The project was implemented using CSC cPouta virtual machines, which were accessed both at school and remotely. Work was conducted on-site until Christmas, after which most of the tasks were carried out remotely. Communication during the project was handled through the Discord chat application and its dedicated servers. After the initial familiarization phase, Task Force team members specialized in different areas and topics.

Performance testing results were obtained through load testing. It was observed that the performance of the PrestaShop e-commerce platform could be improved. This was determined by analyzing the results of the load tests, specifically response times, percentiles, pass rates, and time limits.

The conclusion is that there is room for improvement in the performance of PrestaShop. Potential improvements include increasing the resources of the virtual machines and enhancing scalability. Load test results showed that performance deteriorates with high traffic.

Keywords/tags (subjects)

Testing, Performance Testing, Grafana, Prometheus, K6, PrestaShop

Miscellaneous (Confidential information)

Sisältö

Kuviot

Taulukot

1 Johdanto

1.1 Tausta ja toimeksiantaja

Työn toimeksiantajana toimi Jyväskylän ammattikorkeakoulu (JAMK). Joka perustettiin 1994 Keski-Suomeen, Jyväskylän kaupunkiin. JAMK:ssa oli noin 9500 opiskelijaa, joista vuosittain valmistui noin 1500. Valmistuneista opiskelijoista jopa 80 % oli töissä vuosi valmistumisen jälkeen. JAMK oli osana Eurooppa-yliopisto E³UDRES²:ia ja ottaa myös opiskelijoita yli 70 eri maasta (JAMK, N.d.).

Saatiin mahdollisuus työskennellä Jyväskylän IT instituutin hankkeessa, jota johti Marko Rintamäki. Hankkeen tarkoitus oli luoda Task Force-tiimi, joka harjoittaa omalle alalleen sopivia taitoja, kuten tietoturvaa, testaamista tai automaatiota. Lisäksi tiimi teki materiaaleja, joita hyödynnetään osana Future Factory (FF) toteutusta, sekä tulevaisuudessa osana WIMMA Capstonea. Materiaalien lisäksi Task Force toimi tukijoukkona FF osallistuville opiskelijoille, jos he tarvitsivat apua jossakin.

Future Factory-projekti oli kurssi, jossa opiskelijat saivat hankkia valmiuksia suunnitella ja toteuttaa työelämän kehitysprojekteja. Osallistujat jaettiin tiimeihin, joissa he pääsivät soveltamaan oman alansa osaamista työelämän monimuotoisiin kehittämisprosesseihin. Tarkoituksena oli, että opintojakson suoritettua osattiin tunnistaa ja toteuttaa työelämän kehittämishankkeita sekä toimia vastuullisesti ja asiakaslähtöisesti projektiryhmissä. Vuorovaikutuksella ja asiantuntijuudella edistettiin kehittämisprojektien ratkaisua, ja hyödynnettiin kestävän kehityksen periaatteita sekä innovaatio- ja projektitoiminnan menetelmiä.

WIMMA Capstone™ oli projektioppimisympäristö, jossa yritykset, yhteisöt ja opiskelijat kohtasivat työelämän todellisia haasteita. Opiskelijatiimit määrittelivät eri ohjelmistoyritysten tai yhteisöjen asettamia tuotekehityshaasteita. Työ tehtiin yhteistyössä ohjaajien, vertaisohjaajien ja yritysedustajien kanssa. Projektissa keskityttiin tehokkaaseen tiimityöskentelyyn, mutta myös aktiivisesti opitun jakamiseen tiimirajojen yli. Projektitehtävät nousivat asiakkaiden tarpeista, mutta niiden tavoitteet muokattiin opiskelijoille merkitykselliseksi oppimiskokonaisuudeksi (WIMMA Capstone™, N.d.).

WIMMA Capstone™ tavoitteena oli luoda saumaton yhteys koulun ja työelämän välille. Projektimuotoinen opiskelu mahdollisti uuden tiedon ja hyödyllisten verkostojen muodostumisen luonnollisena osana opiskelijan opintopolkua.

Projektityöskentelyä tehtiin tiiviisti ryhmässä noin kolmen kuukauden ajan. WIMMA Capstone käynnistettiin pienimuotoisesti vuonna 2025, ja sitä laajennettiin tulevina vuosina (WIMMA Capstone™, N.d).

1.2 Tavoite ja tehtävät

Opinnäytetyön tavoite oli tutustua, tehdä ja dokumentoida suorituskykytestausta k6 ohjelmalla, kun kohteena on Prestashop nettikauppa-alusta. Tehtäviä opinnäytetyössä olivat testata pystytetyn Prestashop kauppa-alustan suorituskykyä ja dokumentoida siitä esimerkki. Käytännön työhön kuului testikohteen sekä testausohjelman pystytys, muiden tarvittavien ohjelmien asennus, testausprosessi, testidatan keruu, visualisointi ja analysointi.

Tekninen työ tullaan suorittamaan CSC:n virtuaalikoneilla. CSC on suomalainen voittoa tavoittelematon erityistehtäväyhtiö (Suomen valtio omistaa 70 % ja korkeakoulut 30 %), joka tarjoaa työkaluja tieteelliseen laskentaan ja datan hallintaan (Mikä CSC, N.d.).

Suorituskykytestaus, jota kutsutaan joskus myös perf-testaukseksi, on menetelmä, jolla arvioidaan, miten hyvin jokin tuote suoriutuu erilaisista kuormitustilanteista. Tämä voi tarkoittaa esimerkiksi verkkosivuston tai ohjelmiston tehokkuuden mittaamista riippuen siitä, mitä tuotetta testataan (ZAPTEST, N.d.).

Tällaisen testauksen päätavoitteena on tunnistaa ne tekijät, jotka voivat heikentää tuotteen toimintaa. Näiden ongelmakohtien tunnistaminen varhaisessa vaiheessa tuotteen kehitystä auttaa ehkäisemään suurempia ongelmia myöhemmin. Näitä kriittisiä kohtia kutsutaan usein pullonkauloiksi, ja ne ovat yksittäisiä osia järjestelmässä, jotka hidastavat koko ohjelmiston toimintaa (ZAPTEST, N.d.).

1.3 Tutkimuksellinen kehitystyö

Tutkimuksen ja kehittämistoiminnan risteyspaikkaa voidaan lähestyä sekä kehittämistoiminnan että tutkimuksen suunnasta. Ensinnäkin voidaan puhua kehittävästä tutkimuksesta. Siinä ajattelun logiikka kulkee tutkimuksellista kysymyksenasetteluista ja metodologisista tarkasteluista kohti konkreettista kehittämistoimintaa. Tietoa tuotetaan käytännön kehittämisprosessien yhteydessä, mutta tiedeyhteisön intressin mukaisesti. Pääpaino on sanalla tutkimus, mutta sen suunta on kehittämisessä (Toikko & Rantanen, 2009, 21–22).

Toiseksi voidaan puhua tutkimuksellisesta kehittämistoiminnasta, jolloin käytännön ongelmat ja kysymykset ohjaavat tiedontuotantoa. Tietoa tuotetaan aidoissa käytännön toimintaympäristöissä ja tutkimukselliset asetelmat ja menetelmät toimivat apuna tässä. Tällöin voidaan korostaa kehittämistoiminnan tutkimuksellista luonnetta. Pääpaino on sanalla kehittämistoiminta, mutta siinä pyritään hyödyntämään tutkimuksellisia periaatteita. Konkreettinen kehittämistoiminta määrittelee tutkimuksen reunaehdot, joten tutkimusasetelmat ovat kehittämistoiminnalle alisteisessa asemassa (Toikko & Rantanen, 2009, 22).

2 Suorituskykytestaus

2.1 Yleistä

Suorituskykytestaus oli käytäntö, jolla arvioitiin, kuinka järjestelmä toimii reagoivuuden ja vakauden suhteen tietyssä työkuormassa. Suorituskykytestejä suoritetiin yleensä nopeuden, kestävyyden, luotettavuuden ja sovelluksen koon tutkimiseksi. Sovellusten käyttöönotto, menestys ja tuottavuus riippuivat suoraan suorituskyvyn testauksen asianmukaisesta toteutuksesta (Tricentis Staff, 2024).

Mit Thakkar (N.d) kertoi blogissaan että, suorituskykytestaus mittaa sovelluksen vasteaikaa, nopeutta, käytettävyyttä ja vakautta tietyllä kuormitustasolla. Kukaan käyttäjä ei haluaisi käyttää sovelluksia, jotka reagoivat hitaasti, sisältävät navigointiongelmia tai monimutkaisia kojelautoja jne. Siksi suorituskykytestauksen priorisointi koko SDLC-prosessin (Software Development Life Cycle) ajan on välttämätöntä.

Mit Thakkar (N.d) myös kirjoitti, että suorituskykytestauksen merkitys oli tärkeä. Sillä voitiin seurata ja korjata suorituskykyvirheitä, jotka saattoivat vaikuttaa negatiivisesti käyttäjäkokemukseen. Testaus oli myös tärkeää vakauden, luotettavuuden ja nopeuden analysoimiseksi. Se auttaa arvioimaan ohjelmiston kapasiteettia ja kykyä käsitellä erilaisia kuormia. Ohjelmiston suorituskykytestauksen merkitys oli olennainen, koska se oli välttämätön vaihe kaikissa SDLC-projekteissa ja varmistaa, että järjestelmä oli tehokas ja luotettava.

Suorituskykytestauksen yleisimpiä testityyppejä olivat rasitustestaus (Stress testing) ja kuormitustestaus (Load testing). Rasitustestauksen tarkoituksena oli löytää ja ymmärtää järjestelmän kapasiteetin ylärajat. Sovellukseen kohdistettiin äärimmäinen kuormitus sen vankkuuden selvittämiseksi. Lisäksi suoritettiin myös kestotestausta (Soak testing), joka tunnettiin myös nimellä kestävyystestaus (Endurance testing). Tämän testauksen tavoitteena oli arvioida, pystyykö järjestelmä ylläpitämään jatkuvan odotetun kuorman. Mahdolliset muistivuodot havaittiin seuraamalla muistin käyttöä (Hooda & Chhillar 2015, 12).

Kestotestaus varmisti, että läpimenoteho ja vasteaika pitkän jatkuvan käytön jälkeen olivat yhtä hyviä tai parempia kuin testin alussa. Sen päätavoitteena oli selvittää järjestelmän käyttäytyminen jatkuvassa käytössä. Järjestelmän tehokkuuden ja vankkuuden arvioimiseksi lasketyiin keskimääräinen korjausaika (MTTR) ja keskimääräinen aika vikojen välillä (MTBF) (Hooda & Chhillar 2015, 12).

Hooda & Chhillar (2015, 12) kirjoittavat, että piikkitestaus (Spike testing) tehdään lisäämällä äkillisesti käyttäjämäärää tai käyttäjien tuottamaa kuormitusta merkittävästi ja tarkkailemalla järjestelmän käyttäytymistä. Ohjelmiston suorituskyvyn testaamiseen käytettiin pääasiassa työkaluja, sillä kuorman testaaminen manuaalisesti olisi erittäin vaikeaa. Saatavilla oli useita ilmaisia työkaluja, kuten SoapUI ja JMeter, joita käytetään ohjelmistojen suorituskykytestaukseen. Yksi suosituimmista työkaluista oli LoadRunner, ja lisäksi IBM tarjoaa useita suorituskykytestaukseen tarkoitettuja työkaluja.

2.2 Suorituskykytestaus tyyppejä

2.2.1 Kuormitustestaus (Load)

Kuormitustestauksella (eng. Load testing) voitiin tarkoittaa yleistermiä, joka sisältää muita suorituskykytestauksen tyyppejä, kuten stressi- tai piikkitestaus. Kuormitustestauksella voitiin myös tarkoitta suorituskykytestaus tyyppiä, jossa simuloidaan sovellukseen tai järjestelmään kohdistuvaa kuormitusta suorituskyvyn arvioimiseksi.

Kuormitustestauksen tarkoituksena oli arvioida, kuinka järjestelmä tai sovellus toimii sille tyypillisellä kuormituksella. Tyypillisellä kuormituksella voitiin tarkoittaa tavallista tuotantopäivää tai keskimääräistä liikennettä tietyllä aikavälillä.

Kuormitustesteissä simuloitiin yhtäaikaisten käyttäjien ja sekunnissa tehtyjen pyyntöjen määrää, jotka vastasivat tuotantoympäristön keskimääräistä käyttöä. Tyypillisesti testissä kasvatettiin läpimenoa (throughput) tai virtuaalikäyttäjien (VU, virtual users) määrää asteittain ja pidettiin kuorma vakaana jonkin aikaa. Järjestelmän ominaisuuksista riippuen testi voitiin päättää äkillisesti tai sisältää lyhyen kuorman vähennysvaiheen (ramp-down) (Types of load testing N.d.).

2.2.2 Piikkitestaus (Spike)

Piikkitestauksen (eng. Spike testing) tarkoituksena oli katsoa, selviääkö järjestelmä tai sovellus äkillisistä ja massiivisista käyttöpiikeistä sekä toimiiko se niiden aikana oikein.

Piikkitestit olivat hyödyllisiä tilanteissa, joissa järjestelmä pystyi kohdata poikkeuksellisen suuria liikennemääriä. Esimerkkejä tällaisista tapahtumista olivat lippujen myynti (esim. Taylor Swift -konsertit), tuotelanseeraukset (kuten PS5), mainoskampanjat (Super Bowl -mainokset), määräaikojen lähestyminen (veroilmoituksen jättöpäivä) ja sesonkialet (Black Friday). Liikennepiikkejä voi aiheutua myös toistuvammista tapahtumista, kuten ruuhkatunneista.

Piikkitestauksessa kuorma nostettiin erittäin korkeaksi hyvin lyhyessä ajassa tai ilman asteittaista kasvua (ramp-up). Samoin kuormaa voitiin vähentää (ramp-down) hyvin nopeasti tai ei lainkaan, jolloin testi toistetaan vain kerran.

Tämä testityyppi voi sisältää erilaisia prosesseja kuin muut suorituskykytestit, koska piikit eivät ole osa järjestelmän tyypillistä päivittäistä käyttöä. Testiskriptiä voitiin joutua muokkaamaan lisäämällä, poistamalla tai muuttamalla prosesseja, joita ei yleensä sisällytetä keskimääräisen kuorman testeihin (Types of load testing N.d.).

2.2.3 Kestotestaus (Soak)

Kestotestauksessa (eng. Soak testing) oli kuormitustestiä muistuttava testi, jotka eroavat toisistaan testin pituuden puolesta. Kestotestissä kuorma oli keskimääräisellä tasolla, mutta testi itsessään kestää useita tunteja tai jopa päiviä. Vaikka kestotestaus kestääkin pidempään kuin kuormitustestaus, sen kuorman nosto- ja laskemisvaiheet (ramp-up ja ramp-down) ovat samankaltaisia kuin kuormitustestauksessa.

Kestotestin tavoitteena oli järjestelmän tai sovelluksen suorituskyvyn heikkenemisen ja resurssien kulutuksen analysointi pitkällä aikavälillä. Oli tärkeää tietää, jos sovellus kaatuu muutaman tunnin käytön jälkeen, sillä heikkouksien havaitseminen ja korjaaminen alussa oli halvempaa ja helpompaa kuin sen tekeminen projektin loppuvaiheessa. Kestotestauksessa myös seurattiin järjestelmän tai sovelluksen saatavuutta sekä vakautta.

2.2.4 Stressitestaus (Stress)

Stressitestaus (eng. Stress testing) on muistuttaa kuormitustestaus, mutta erona oli, että stressitestauksessa järjestelmään tai sovellukseen kohdistuva kuorma oli suurempi ja testi voi kestää hiukan pidempään. Stressitestauksen tarkoituksena ei ole löytää järjestelmän tai sovelluksen rikkoutumisrajaa, vaan katsoa miten kohde selvisi keskimääräistä kovemman kuorman alla. Näin voidaan selvittää järjestelmän tai sovelluksen vakautta ja luotettavuutta.

2.3 Työkalut

2.3.1 K6

K6 oli avoime lähdekoodin kuormitus- ja suorituskykytestaustyökalu, joka sai alkunsa vuonna 2000. Tuolloin tulevan työkalun tekijät työskentelivät massiivisen moninpeli online-roolipelin kanssa. Perustajatiimi huomasi tarpeen aikaisen vaiheen kuormitustestaukseen, ja päättivät avata

konsulttitoimiston nimeltä LoadImpact. He tekivät konsultointia organisaatioille, muun muassa Euroopan Avaruusjärjestölle.

Vuonna 2009 LoadImpact päätti lopettaa konsultointityöt ja vaihtoi toimintamalliaan SaaS (Software as a Service). Tämä pilviohjainen ohjelmistotoimitusmalli mahdollisti pilvipohjaisen kuormitustestaustyökalun kehityksen, joka sai yrityksensä nimen, LoadImpact.

Kuormitustestaustyökalu, joka tultaisiin tuntemaan nimellä K6, alettiin kehittämään vuonna 2016. K6 kehitys jatkui ja vuonna 2020 julkaistiin xK6 (K6-extensions). XK6 mahdollisti käyttäjien luoda omia K6-binäärejä, jotka sisältivät haluttuja laajennuksia. Näin käyttäjät saivat niputettua haluttuja ominaisuuksia ja toimintoja, mitkä eivät kuuluneet K6:n ydintoimintoihin.

2.3.2 Prometheus

Rajesh Kumar (2021) kirjoitti blogissaan, että Prometheus oli ilmainen, avoimenlähdekoodin järjestelmänmonitorointi- ja hälytystyökalu. Sen kehitys alkoi 2012 SoundCloud:in toimesta ja vuonna 2016 julkaistiin Prometheus versio 1.0. Seuraavan vuonna Prometheus päivitettiin versioon 2.0 (Kumar, R. 2021). 2018 se valmistui Cloud Native Computing Foundation (CNCF) hautomosta (Grafana OSS and Enterprise. N.d).

Sekä Kumar (2021) että Hussain (2024) olivat samaa mieltä siitä, että Prometheus sisältää hyviä ominaisuuksia. Näitä ominaisuuksia olivat muun muassa moniulotteinen datamalli, ei riippuvuutta hajautetusta tallennuksesta, pull-malli metriikka haku ja kohteiden löytäminen (static config ja service discovery) (Kumar, R. 2021.; Hussain, S. 2024).

Monipuolinen datamalli tarkoittaa sitä, miten Prometheus tallentaa keräämänsä metriikat. Ne on tallennettu aikasarjoihin (time series) jotka koostuvat kahdesta osasta, metriikan nimestä ja avain/arvo-pareista. Tämä mahdollistaa tarkan ja joustavan analyysin, jolla voi suodattaa, ryhmitellä ja yhdistellä dataa tehokkaasti (What is Prometheus. N.d).

Prometheuksen toimii itsenäisenä palvelimena, eli se ei tarvitse klusteria tai ulkoista tietokantaa datan tallentamiseen. Tämä yksinkertaistaa asennusta ja hallintaa. Myös luotettavuus kasvaa, sillä

jos yksi node toimii, niin prometheus kerää dataa normaalisti. Jos kuitenkin tarvittaisiin pitkän aikavälin tallennusta tai korkeaa käytettävyyttä, voi Prometheuksen yhdistää muihin tallennus ratkaisuihin, kuten Thanos tai Cortex (What is Prometheus. N.d).

Prometheus käyttää Push:in sijasta Scrape metriikka hakua, joka Prometheus kontekstissa tarkoittaa samaa kuin Pull. Tämä tarkoittaa sitä, että Prometheus keräsi kohteesta metriikoita ”raapimalla” sen http-päätepisteitä. Pull valittiin Push:in ylitse, koska sillä voi käydä manuaalisesti selaimella tarkastamassa kohteen tilan ja jos kohde menee alas, sen havaitseminen oli helpompaa ja nopeampaa. Scrape mahdollistaa myös käynnistää lisäseurantainstansseja tarpeen mukaan, vaikka omalla kannettavalla (What is Prometheus. N.d).

Kohteet, joista Prometheus raapii metriikoita, voitiin asettaa kahdella tapaa, staattisesti tai palveluntunnistuksella. Palvelun löytäminen staattisesti tapahtui siten, että seurattavan palvelun tarvittavat tiedot laitetaan Prometheuksen konfigurointi tiedostoon, jossa oli kohta scrape kohteille (What is Prometheus. N.d). Palveluntunnistus löytää ja tunnistaa ympäristössä olevat kohteet automaattisesti. Tämä oli hyvä vaihtoehto staattiseen verrattuna, jos kyseessä on mikropalvelu arkkitehtuuri, jolloin muutoksia tapahtuu jatkuvasti (Harsh. 2024).

Prometheus pystyi kerätä metriikoita itsestään mutta, jotta se voisi kerätä niitä muualta, se vaatii agentin. Tämä agentti tulee keräämään halutusta kohteesta järjestelmädataa ja, muuttaa sen muotoa Prometheukselle sopivaksi. Node exporter oli hyvä vaihtoehto tähän tarkoitukseen, sillä se oli Prometheuksen alaisuudessa ylläpidetty projekti (What is Prometheus. N.d.; Prometheus Node Exporter Setup. N.d).

2.3.3 Grafana

Grafana oli avoimen lähdekoodin visualisointi- ja analytiikkatyökalu. Se mahdollistaa metriikkojen ja lokie seurannan, tutkimisen ja esittämisen eri tietolähteistä. Tietolähteet voivat sijaita eri paikoissa, ne pystyttiin silti löytämään. Grafanalla voi myös laittaa hälytyksiä ja kyselyitä, sekä tehdä koontinäyttöjä ihmisille ymmärrettävässä muodossa. Grafanasta oli olemassa ilmainen, yhteisön ylläpitämä versio sekä kaupallinen, yrityskäyttöön tarkoitettu Grafana Enterprise, jossa oli lisäominaisuuksia, 27/4 tuki sekä koulutustarjontaa (Slingerland, C. 2024).

Grafanalla oli myös muita ohjelmistoja kuten Grafana Loki, Mimir ja Cloud. Grafana Loki oli avoimen lähdekoodin komponenttisarja, josta voidaan tehdä toimiva lokitusratkaisu. Toisin kuten muut lokitusjärjestelmät, jotka käsittelevät itse lokitietoja, Loki-ohjelmiston indeksoi kohteesta vain metatietoja sekä tunnisteita. Itse lokitieto pakattiin ja tallennettiin lohkoihin pilvipalveluun tai paikalliselle järjestelmälle (Slingerland, C. 2024).

Samoin kuin Loki, Grafana Mimir oli avoimen lähdekoodin ohjelmistoprojekti. Se on tallennusratkaisu, joka tarjosi vaakasuunnassa skaalautuvaa, erittäin saatavilla olevaa, monivuokrakäyttöistä ja pitkän aikavälin tallennusta metriikoille (Grafana OSS and Enterprise. N.d). Vaakatason skaalautuvuus tarkoitti sitä, että järjestelmään lisätään solmuja tai koneita, uusien vaatimusten täyttämiseksi. Vertikaali skaalauksessa ei lisätä solmuja tai koneita, vaan annetaan järjestelmälle lisää tehoa. Toisin sanoen vaakatason skaalauksessa käytössä oli paljon pieniä järjestelmiä ja vertikaalisessa oli yksi iso (Slingerland, C. 2024).

Korkea saatavuus tarkoittaa järjestelmän kykyä olla luotettava ja saatavissa lähes koko ajan. Tämä tarkoittaa sitä, että järjestelmän oli kestettävä käyttökatkokset, suunnitellut seisokit sekä sivunkattavat onnettomuudet ja hyökkäykset. Korkea saatavuus antaa järjestelmälle joustavuutta, koska työlle ei tule katkoksia ja lisää tietoturvallisuutta, sillä tietoturvajärjestelmä on aina päällä suojaamassa dataa (Flinders, M. & Smalley, I. 2024). Monivuokrakäyttö oli arkkitehtuuriratkaisu, jossa yhtä ohjelmistoinstanssi palvelee useampaa vuokralaista (käyttäjää), samanaikaisesti. Tämä tarkoittaa sitä, että yksi ohjelmainstanssi pyörii palvelimella, joka sitten palveli useita käyttäjiä. Monivuokrakäyttö soveltuu hyvin pilvipalveluihin, ja onkin saanut niiden keskuudessa suosiota. Etuja monivuokrakäytöllä muihin arkkitehtuuriratkaisuihin olivat muun muassa, hinta, maksumalli, helppokäyttöisyys asiakkaan puolelta ja skaalautuvuus (Bigelow, J.S. & Gillis, S.A. 2024).

Koska koneita oli vähemmän kuin perinteisessä arkkitehtuurissa, monikäyttöarkkitehtuuri on halvempi. Monikäyttöarkkitehtuuri-käyttöiset pilvipalvelut myös usein käyttävät PAYG eli Pay-As-You-Go ja Pay-For-What-You-Use malleja, joissa maksetaan käytön perusteella. Monivuokrakäyttö on myös asiakkaisen puolesta helpompaa, sillä heidän ei tarvitse huolehtia koneisto tai päivityksistä, koska ne olivat isännöitsijän vastuulla. Ratkaisu oli myös helposti skaalautuva, sillä se on suunni-

teltu kestämään nousevia käyttäjä- ja datamääriä, ja koska resurssit ovat jaettu, uusien vuokralaisten lisääminen vie yleensä vähemmän resursseja verrattuna yksivuokralais-arkkitehtuuriin (Bigelow, J.S. & Gillis, S.A. 2024).

2.4 Prestashop

PrestaShop oli avoimen lähdekoodin verkkokauppa-alusta, ja oli perustoiminnoiltaan ilmainen. Ensimmäinen vakaa versio PrestaShopisa julkaisitiin vuoden 2008 alkupuolella. BuiltWith:in mukaan PrestaShop oli miljoonasta suosituimmista sivustosta viidenneksi yleisin verkkokauppa-alusta, noin 0,5 %. Kaikista maailman verkkokaupoista PrestaShop toteuttaa noin 3 % (Kataja, J. 2016.; eCommerce Uasge Distribution in the Top 1 Million Sites, N.d.).

3 K6 suorituskykytestauksen toteutus

3.1 Ympäristö

Ympäristö, jossa opinnäytetyön käytännön toteutus suoritetaan, on CSC:n tarjoama cPouta-palvelu. Pouta-palveluja CSC:llä on cPouta ja ePouta, jotka olivat kumpiki Infrastructure-as-a-Service (IaaS) palveluita. Pouta on yleistermi kummallekin palvelulle.

EPouta eroaa cPoudasta siinä, että ePouta oli yksityinen pilvi suunniteltu saavuttamaan salassapidettävien tietojen turvallisuusvaatimukset. CPouta puolestaan oli julkinen pilvi, josta oli helppo muodostaa yhteys internettiin. CPoutaan voi hakea pääsyä itsepalveluna, kun taas ePouta haku prosessi oli vaativampi ja siinä saattoi kestää viikkoja.

Käytännön työn tavoitteena on pystyttää cPoutaan kaksi virtuaalikonetta, testaus kone (K6) ja kohde kone (Prestashop). Testaus kone tuli sisältämään K6-testausohjelman, Prometheus monitorointiohjelman sekä Grafanan, jolla visualisoitiin raavittu data. Kohde koneeseen asennetaan Docker-compose, jolla taas asennettiin Prestashop, käyttäen Task Force tiimin valitsemaa versiota 1.7.8.0. Samalla asennettiin myös muita ohjelmia, joita Prestashopin käyttö edellyttää kuten mySQL. Testi koneelle asennettu Prometheus käsittelee tietynlaista dataa. Kohde koneesta haluttu data tultiin keräämään Node Exporter agentilla, joka paljastaa järjestelmä tason metriikoita muodossa, jota Prometheus voi raapia ja ymmärtää.

Valmiissa ympäristössä oli siis kaksi virtuaalikonetta. Toisessa virtuaalikoneessa on K6 testausohjelma, Prometheus monitorointiohjelma, sekä Grafana datan visualisoimiseksi. Prometheus raapii dataa toisesta koneesta, jossa pyörii Node Exporter agentti, joka keräsi dataa järjestelmän suorituskyvystä muodossa, jota Prometheus ymmärtää. Tässä koneessa pyörii myös Prestashop verkkokauppa-alusta.

Prestashopin ollessa ylhäällä, tehdään K6 ohjelmalla testejä, jotka luovat liikennettä Prestashop kauppaan virtuaalikäyttäjillä. Tämä liike voidaan todeta Grafanasta, jossa Node Exporterin ja Prometheuksen keräämä data näkyy visuaalisesti.

Kuvio 1. Grafanan kojelauta

3.2 Asennukset

3.2.1 K6 ja Prestashop

K6 asentaminen oli hyvin yksinkertaista. Piti lisätä K6 virallinen ohjelmalähde ja GPG-allekirjoitus. Kun ne oli tehty, päivitettiin kirjasto ja asennettiin K6 apt-get komennolla.

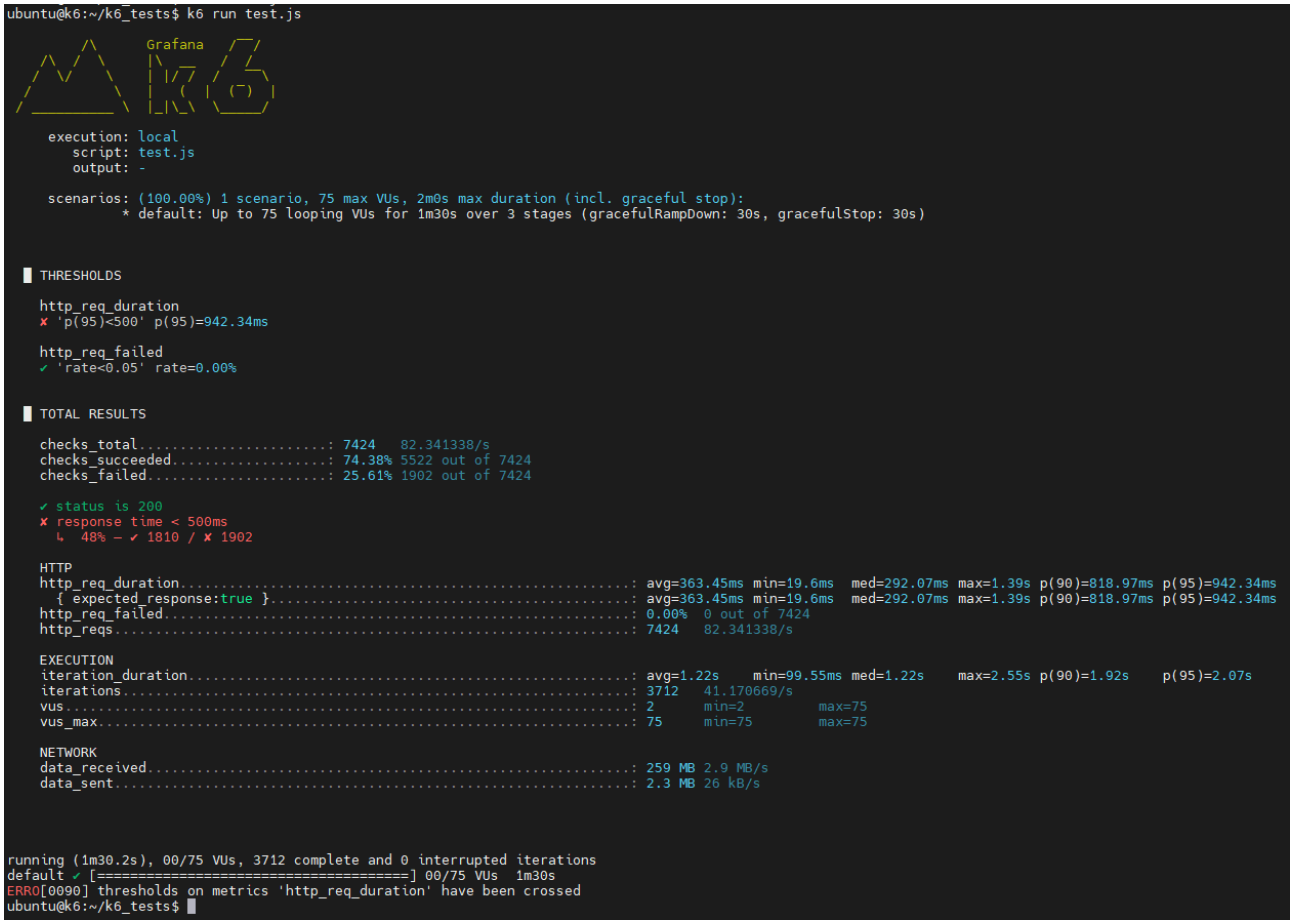
```
ubuntu@k6:~$ sudo gpg -k
sudo gpg --no-default-keyring --keyring /usr/share/keyrings/k6-archive-keyring.gpg --keyserver hkp://keyserver.ubuntu.com:80 --recv-keys C5AD17C747E3415A3642D57D77C6C491D6AC1D69
echo "deb [signed-by=/usr/share/keyrings/k6-archive-keyring.gpg] https://dl.k6.io/deb stable main" | sudo tee /etc/apt/sources.list.d/k6.list
sudo apt-get update
sudo apt-get install k6
```

Kuvio 2. K6 asennus komennot

K6 asentamisen jälkeen testattiin toimiiko se, tekemällä testi.

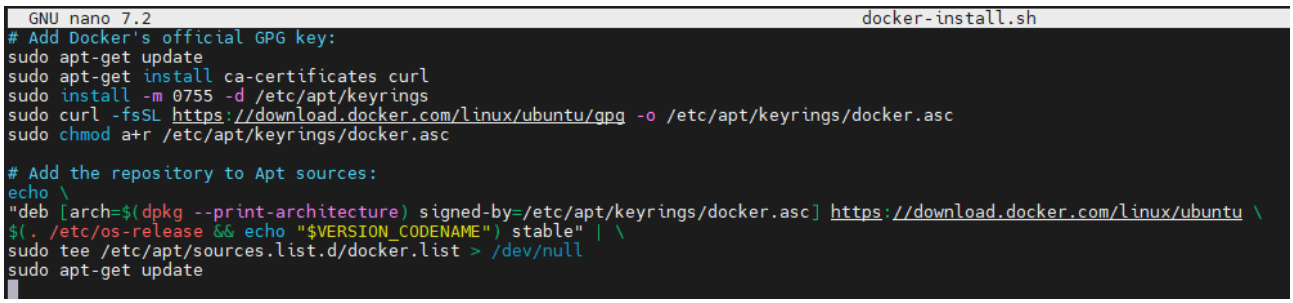
```
  GNU nano 7.2                                                                test.js
import http from 'k6/http';
import { check, sleep  } from 'k6';

export let options = {
  stages:[
    { duration: '30s', target: 75 }, // Ramp-up to 75 virtual users in 30 seconds
    { duration: '30s', target: 75 }, // Hold 150 VUs for 30 seconds
    { duration: '30s', target: 0 }, // Ramp-down to 0 VUs over 30 seconds


  ],
  thresholds: {
    http_req_duration: ['p(95)<500'], // 95% of requests should be < 500ms
    http_req_failed: ['rate<0.05'],   // Error rate should be less than 5%
  },
};

export default function () {
  const res = http.get('https://vm3856.kaj.pouta.csc.fi');
  check(res, {
    'status is 200': (r)=> r.status === 200,
    'response time < 500ms': (r) => r.timings.duration < 500,
  });
   sleep(Math.random() * 1); // Sleep between 0 and 1 seconds
}
```

Kuvio 3. Asennuksen testaus testi



Kuvio 4. Asennuksen testaus testin tulos

Prestashop asennettiin Docker Compose-ohjelmalla, käyttäen Task Force tiimin sisäistä Prestashop versiota 1.7.8.0. Asennus aloitettiin luomalla docker-install.sh asennustiedosto, jonne laitettiin Dockerin virallinen ohjelmalähde sekä GPG-avain. Tämä luotu skripti pystyi hoitamaan Dockerin asennuksen.



Kuvio 5. Docker-install.sh tiedoston sisältö

Docker-install.sh skriptin ajon jälkeen asennettiin Dockerin ydinpalvelut sekä Docker-Compose (työkalu, joka helpottaa usean docker-kontin hallintaa).

```
ubuntu@prestashop:~$ sudo docker version
Client: Docker Engine - Community
 Version:           28.1.1
 API version:       1.49
 Go version:        go1.23.8
 Git commit:        4eba377
 Built:             Fri Apr 18 09:52:14 2025
 OS/Arch:           linux/amd64
 Context:           default

Server: Docker Engine - Community
 Engine:
  Version:          28.1.1
  API version:      1.49 (minimum version 1.24)
  Go version:       go1.23.8
  Git commit:       01f442b
  Built:            Fri Apr 18 09:52:14 2025
  OS/Arch:          linux/amd64
  Experimental:     false
 containerd:
  Version:          1.7.27
  GitCommit:        05044ec0a9a75232cad458027ca83437aae3f4da
 runc:
  Version:          1.2.5
  GitCommit:        v1.2.5-0-g59923ef
 docker-init:
  Version:          0.19.0
  GitCommit:        de40ad0
ubuntu@prestashop:~$
```

Kuvio 6. Docker asennettu

Asennuksien jälkeen ajettiin hello-world kontti, jolla varmistettiin, että Docker toimi oikein.

```
ubuntu@prestashop:~$ sudo docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
e6590344b1a5: Pull complete
Digest: sha256:dd01f97f252193ae3210da231b1dca0cffab4aadb3566692d6730bf93f123a48
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/

ubuntu@prestashop:~$
```

Kuvio 7. Docker hello-world

Dockerin toiminnan varmistuksen jälkeen luotiin docker-compose.yml tiedosto, joka hoiti Prestashop ja MySQL konttien käynnistyksen.

```
  GNU nano 7.2                                                   docker-compose.yml *
# version: '3' returns "the attribute `version` is obsolete" starting with version 25.05

services:
  mysql:  # MySQL service
    container_name: presta-mysql
    image: mysql:5.7
    ports:
      - 3306:3306
    restart: unless-stopped
    environment:
      MYSQL_ROOT_PASSWORD: admin
      MYSQL_DATABASE: prestashop
    networks:
      - prestashop_network_1
    volumes:
      - dbdata:/var/lib/mysql  # Persist database content

  prestashop:  # PrestaShop service
    container_name: prestashop
    image: gitlab.labranet.jamk.fi:4567/presta-shop-development-release-x/presta-shop-service-x:latest  # Latest stable version of PrestaShop from our own repos
    depends_on:
      - mysql
    ports:
      - 80:80  # HTTP
      - 443:443  # HTTPS
    environment:
      DB_SERVER: presta-mysql
      DB_NAME: prestashop
      DB_USER: root
      DB_PASSWD: admin
      PS_FOLDER_ADMIN: admin32sfcxe2  # Renaming admin folder for security
    networks:
      - prestashop_network_1
    volumes:
      - psdata:/var/www/html  # Persist website data
      - apacheconfs:/etc/apache2  # Persist Apache configs and Apache mods
      - apachelogs:/var/log/apache2  # Persist Apache logs
      - ssl:/etc/letsencrypt  # Persist SSL certificates

networks:  # Define the network
  prestashop_network_1:

volumes:  # Define the volumes
  psdata:
  dbdata:
  apacheconfs:
  apachelogs:
  ssl:
```

Kuvio 8. Docker-compose.yml tiedoston sisältö

Yml tiedoston luonnin jälkeen käynnistettiin docker-compose palvelu ja tarkistettiin että palveluille oli luotu kestävä tallennustila (persisten volume).

```
Creating presta-mysql ... done
Creating prestashop   ... done
ubuntu@prestashop:~$
```

Kuvio 9. Prestashop ja MySQL volume luonti

```
CONTAINER ID   IMAGE                                                                                 COMMAND                  CREATED         STATUS         PORTS
                                                   NAMES
701f19d82b1f   gitlab.labranet.jamk.fi:4567/presta-shop-development-release-x/presta-shop-service-x:latest   "docker-php-entrypoi…"   2 minutes ago   Up 2 minutes   0.0.0.0:80->8
0/tcp, [::]:80->80/tcp, 0.0.0.0:443->443/tcp, [::]:443->443/tcp   prestashop
518cd18fdffb   mysql:5.7                                                                             "docker-entrypoint.s…"   2 minutes ago   Up 2 minutes   0.0.0.0:3306-
>3306/tcp, [::]:3306->3306/tcp, 33060/tcp                          presta-mysql
DRIVER    VOLUME NAME
local     ubuntu_apacheconfs
local     ubuntu_apachelogs
local     ubuntu_dbdata
local     ubuntu_psdata
local     ubuntu_ssl
ubuntu@prestashop:~$
```

Kuvio 10. Docker volumet

Kun tallennustilat olivat näkyvissä, mentiin PrestaShop säiliöön. Siellä asennettiin sen vaatimat työkalut (nano sekä systemctl), määritettiin cPouta Domain (dnsutils) sekä luotiin Apache konfiguraatio tiedosto.

```
  GNU nano 5.4                                    /etc/apache2/sites-available/prestashop.conf *
<VirtualHost *:80>
    ServerName vm3856.kaj.pouta.csc.fi
    ServerAdmin aa4231@student.jamk.fi
    DocumentRoot /var/www/html

    <Directory /var/www/html>
        AllowOverride All
        Options +Indexes
        Require all granted
    </Directory>

    ErrorLog /var/log/apache2/prestashop_error.log
    CustomLog /var/log/apache2/prestashop_access.log combined
RewriteEngine on
RewriteCond %{SERVER_NAME} =vm3856.kaj.pouta.csc.fi
RewriteRule ^ https://%{SERVER_NAME}%{REQUEST_URI} [END,NE,R=permanent]
</VirtualHost>
```

Kuvio 11. Apache prestashop.conf tiedoston sisältö

Asennusten ja tiedoston luonnin jälkeen otettiin Apache-moduulit käyttöön, aktivoitiin PrestaShop sivusto, testattiin konfiguraatioita ja käynnistettiin Apache 2 uudelleen.

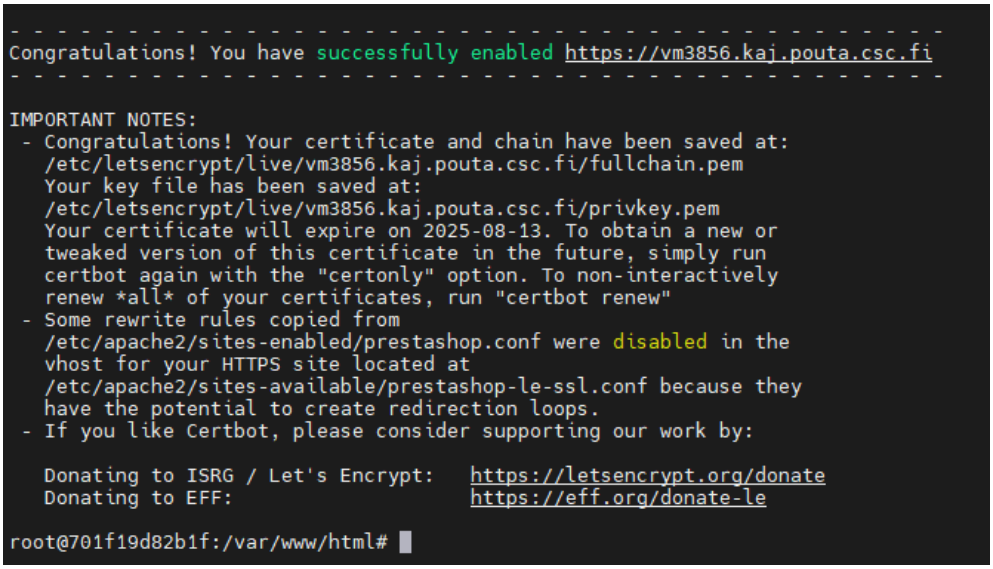
```
root@701f19d82b1f:/var/www/html# a2enmod rewrite headers ssl
Module rewrite already enabled
Enabling module headers.
Considering dependency setenvif for ssl:
Module setenvif already enabled
Considering dependency mime for ssl:
Module mime already enabled
Considering dependency socache_shmcb for ssl:
Enabling module socache_shmcb.
Enabling module ssl.
See /usr/share/doc/apache2/README.Debian.gz on how to configure SSL and create self-signed certificates.
To activate the new configuration, you need to run:
  service apache2 restart
root@701f19d82b1f:/var/www/html#
```

Kuvio 12. Apache-moduulien enablointi

```
root@701f19d82b1f:/var/www/html# a2ensite prestashop.conf
apachectl configtest
systemctl restart apache2
Enabling site prestashop.
To activate the new configuration, you need to run:
  service apache2 reload
AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using 172.18.0.3. Set the 'ServerName' directive globally to suppress this message
Syntax OK
root@701f19d82b1f:/var/www/html#
```

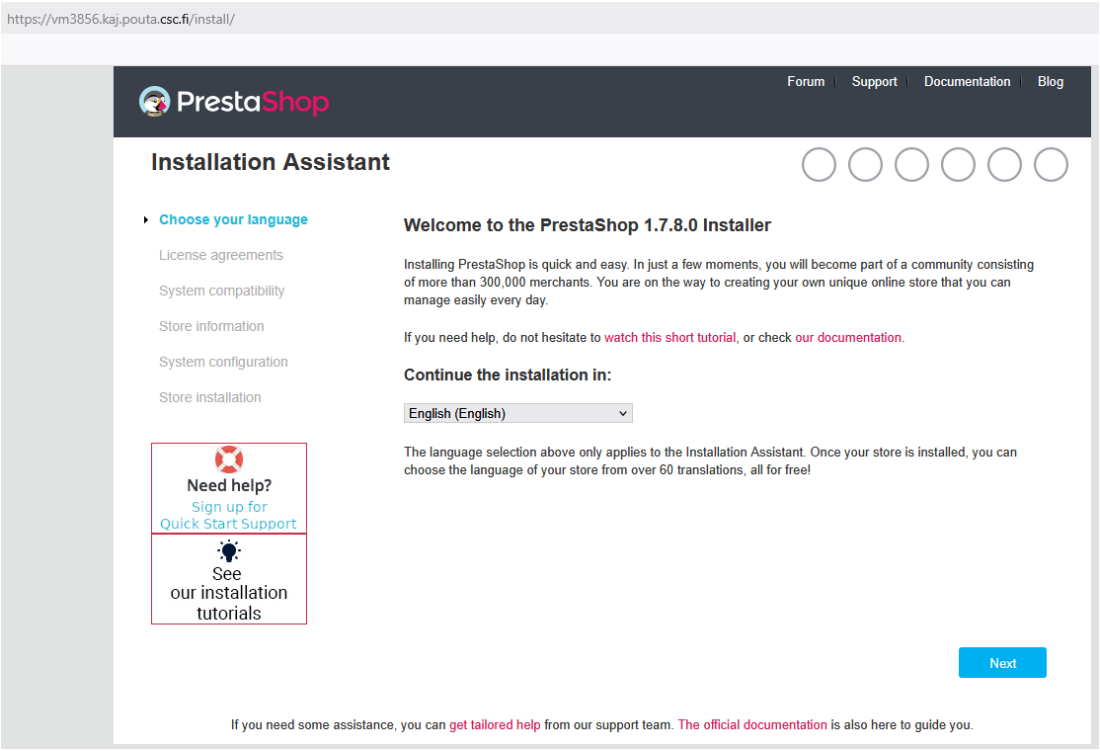
Kuvio 13. PrestaShop sivun aktivointi ja Apache 2 uudelleen käynnistys

Aktivointien jälkeen haettiin SSL-sertifikaatti Certbotilla joka varmisti, että yhteys sivustolle on turvallinen. Lopuksi annettiin Apachelle oikeudet lukea ja kirjoittaa verkkosivun tiedostoja.

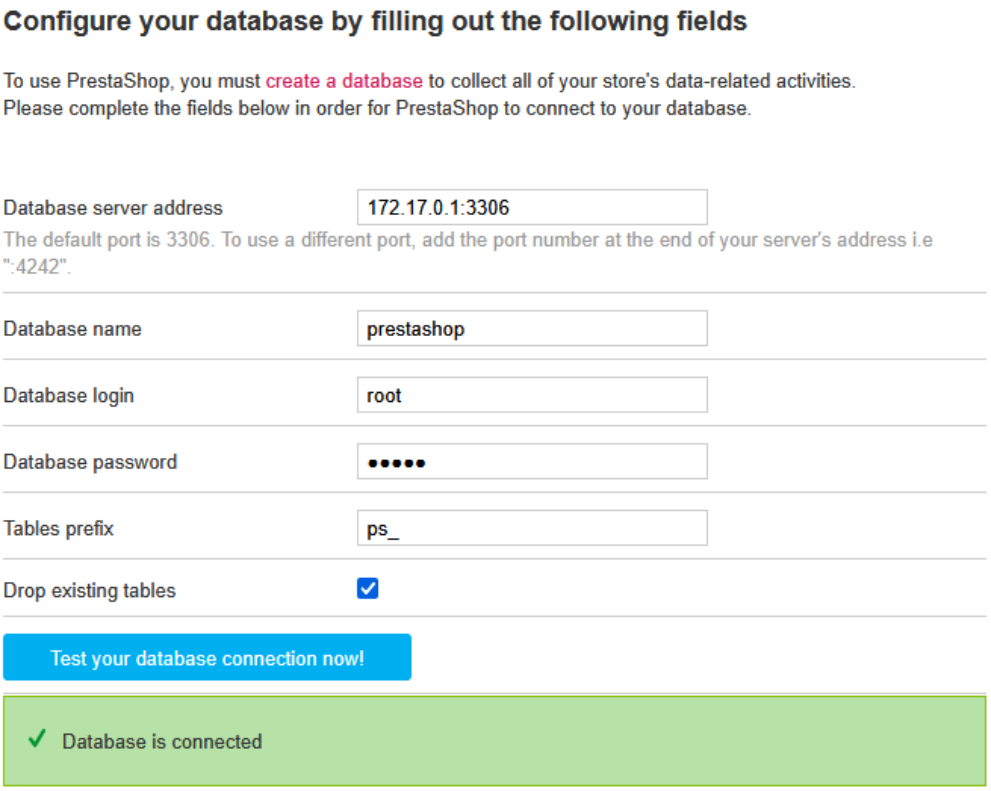


Kuvio 14. SSL-sertifikaatin hankinta onnistui

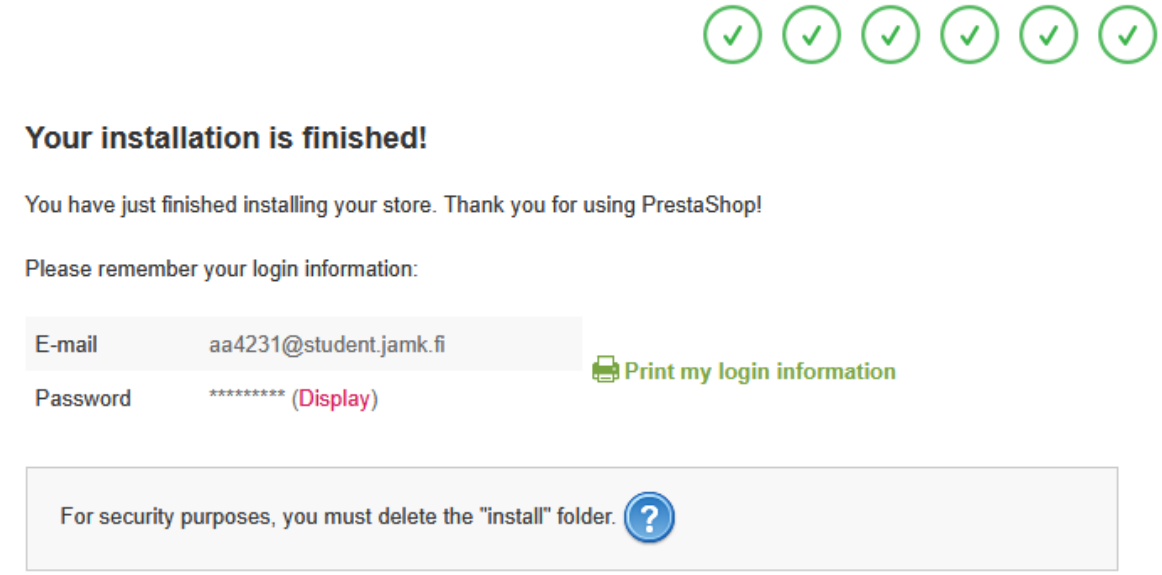
Kun sertifikaatti ja oikeudet oli annettu, voitiin kokeilla toimiiko sivusto. Jos asennusikkuna toimi, niin PrestaShop asennettiin ohjeiden mukaisesti.



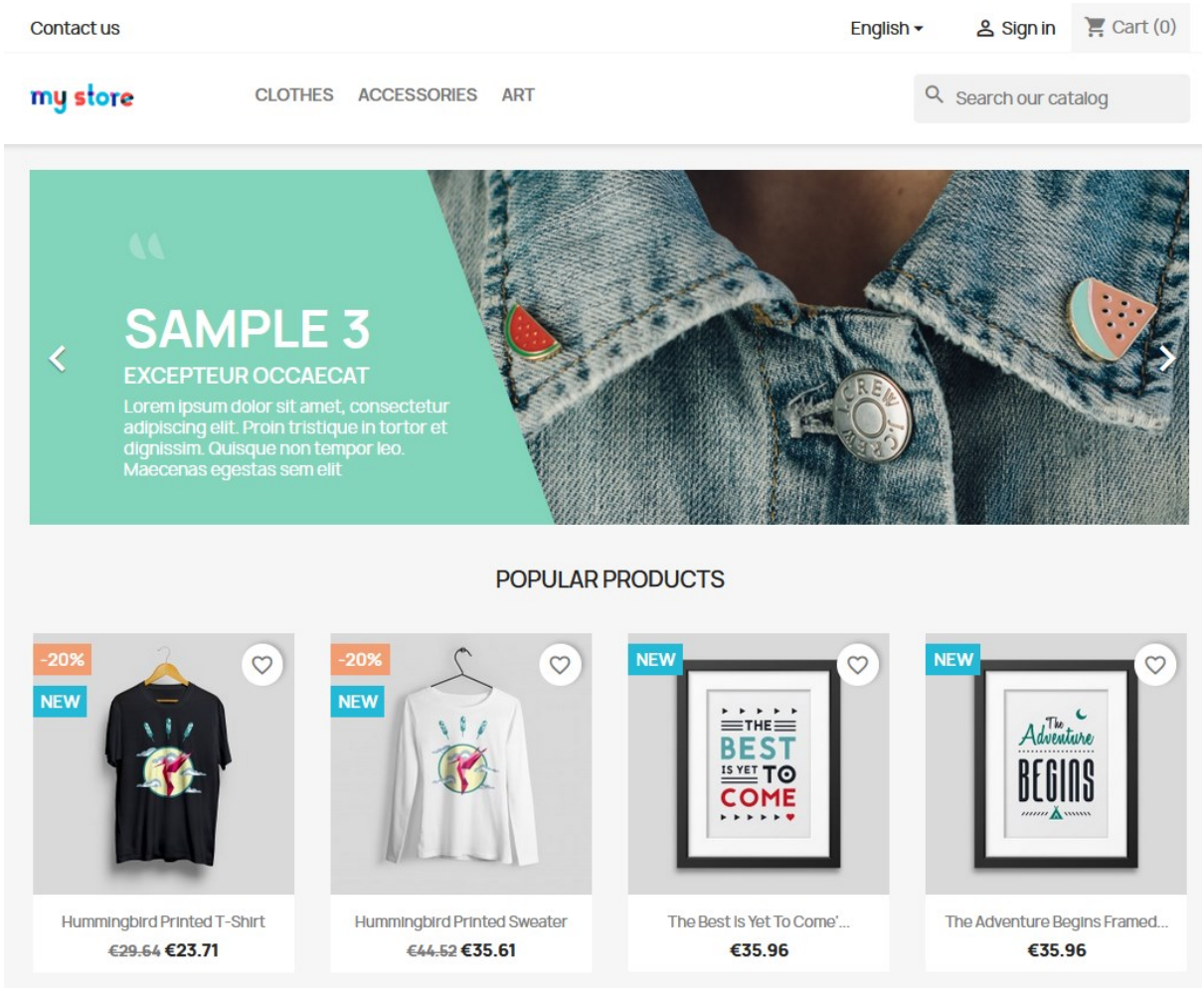
Kuvio 15. PrestaShop asennusikkuna



Kuvio 16. PrestaShop tietokantayhteys toimii



Kuvio 17. PrestaShop asennus valmis

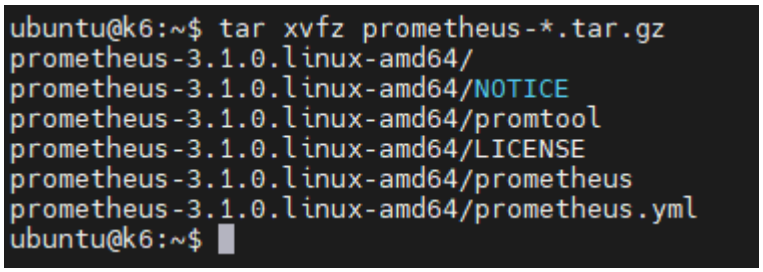


Kuvio 18. PrestaShop etusivu

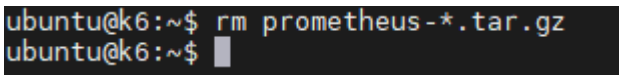
3.2.2 Prometheus

Prometheuksen asennus aloitetaan lataamalla ja purkamalla uusin vakaa versio (3.1.0) GitHubista. Myös portti 9090 on avattava, jotta Prometheuksen käyttöliittymää ja API:a varten.

Kuvio 19. Prometheuksen lataus GitHubista



Kuvio 20. Prometheus.tar.gz purku



Kuvio 21. Poistetaan turha tar.gz

Kun asennus on suoritettu ja asennuspaketti poistettu, luodaan kaksi uutta kansiot /etc/prometheus ja /var/lib/prometheus. /etc/prometheus tulee sisältämään konfiguraatio tiedostoja, /var/lib/prometheus tulee olemaan tallennustila mittaridatalle (Time Series Database, TSDB). Tämän jälkeen siirrytään Prometheus asennukseen ja siirretään sieltä tarvittavat tiedostot järjestelmän oletuspolkuihin.

```
ubuntu@k6:~$ cd prometheus-3.1.0.linux-amd64/
ubuntu@k6:~/prometheus-3.1.0.linux-amd64$
```

Kuvio 22. Siirrytään Prometheus asennuksen sisään

```
ubuntu@k6:~/prometheus-3.1.0.linux-amd64$ sudo mv prometheus promtool /usr/local/bin/
ubuntu@k6:~/prometheus-3.1.0.linux-amd64$
```

Kuvio 23. Siirretään promtool

```
ubuntu@k6:~/prometheus-3.1.0.linux-amd64$ sudo mv prometheus.yml /etc/prometheus/prometheus.yml
ubuntu@k6:~/prometheus-3.1.0.linux-amd64$
```

Kuvio 24. Siirretään prometheus.yml

Tiedostojen ja hakemistojen siirtämisen jälkeen katsotaan, onko Prometheus on asennettu, tarkistamalla sen versio.

```
ubuntu@k6:~/prometheus-3.1.0.linux-amd64$ prometheus --version
prometheus, version 3.1.0 (branch: HEAD, revision: 7086161a93b262aa0949dbf2aba15a5a7b13e0a3)
  build user:       root@74c225e2044f
  build date:       20250102-13:52:43
  go version:       go1.23.4
  platform:         linux/amd64
  tags:             netgo,builtinassets,stringlabels
ubuntu@k6:~/prometheus-3.1.0.linux-amd64$
```

Kuvio 25. Tarkastetaan Prometheus versio

Tarkastuksen jälkeen luodaan Prometheukselle oma rajoitettu järjestelmä käyttäjä. Rajoitettu käyttäjä parantaa turvallisuutta, sillä se estää Prometheusta tekemästä järjestelmätason muutoksia. Käyttäjän luonnin jälkeen asetetaan se aiemmin luotujen kansioiden omistajaksi. Tämä mahdollistaa sen, että Prometheus voi lukea asetustietojaan, sekä kirjoittaa mittausdataa tietokantaansa.

Käyttäjän ja oikeuksien ollessa kunnossa, tehdään systemd tiedosto /etc/systemd/system/prometheus.service. Systemd avulla Prometheukselle voidaan määrittää vikatilanteiden sattuessa auto-

maattinen käynnistys, uudelleenkäynnistys ja hallinta. Tämä automatisointi parantaa ohjelman toimintaa ja nopeuttaa työskentelyä, koska ei tarvitse manuaalisesti käydä käynnistämässä Prometheusta aina uudestaan.

```
ubuntu@k6:~/prometheus-3.1.0.linux-amd64$ sudo nano /etc/systemd/system/prometheus.service
ubuntu@k6:~/prometheus-3.1.0.linux-amd64$
```

Kuvio 26. Luodaan prometheus.service

```
[Unit]
Description=Prometheus
Wants=network-online.target
After=network-online.target

[Service]
User=prometheus
Group=prometheus
Type=simple
Restart=on-failure
RestartSec=5s
ExecStart=/usr/local/bin/prometheus \
    --config.file /etc/prometheus/prometheus.yml \
    --storage.tsdb.path /var/lib/prometheus/ \
    --web.console.templates=/etc/prometheus/consoles \
    --web.console.libraries=/etc/prometheus/console_libraries \
    --web.listen-address=0.0.0.0:9090 \
    --web.enable-lifecycle \
    --log.level=info

[Install]
WantedBy=multi-user.target
```

Kuvio 27. Prometheus.service sisältö

Prometheuksen systemd automaation jälkeen käynnistetään ohjelma uudestaan ja otetaan palvelu käyttöön. Varmistetaan että Prometheus toimii tarkastamalla sen tila.

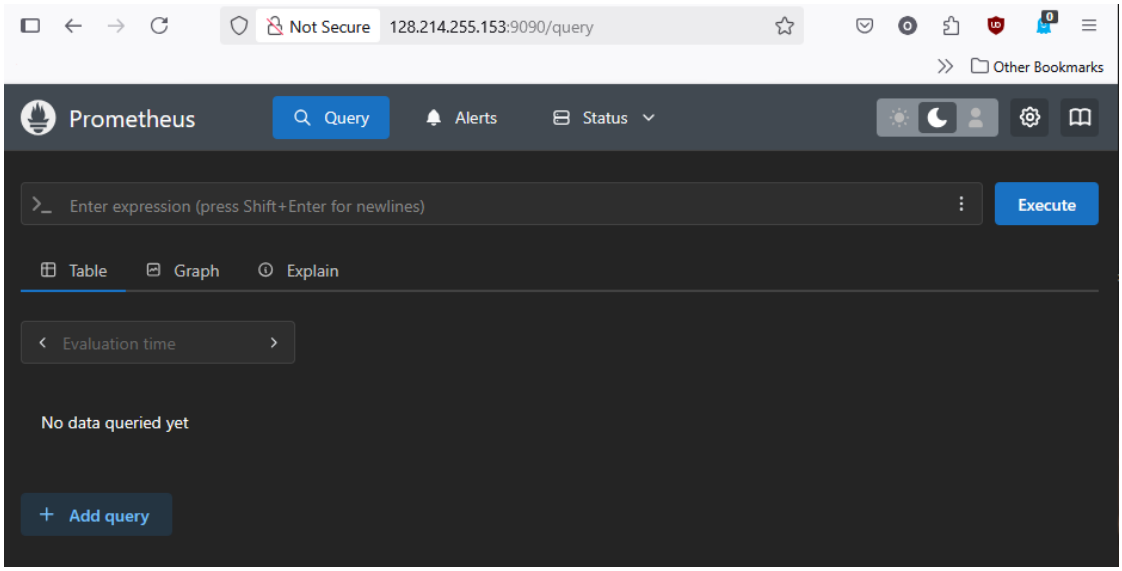
```
ubuntu@k6:~/prometheus-3.1.0.linux-amd64$ sudo systemctl daemon-reload
sudo systemctl enable prometheus
sudo systemctl start prometheus
ubuntu@k6:~/prometheus-3.1.0.linux-amd64$
```

Kuvio 28. Laitetaan Prometheus päälle ja käyttöön

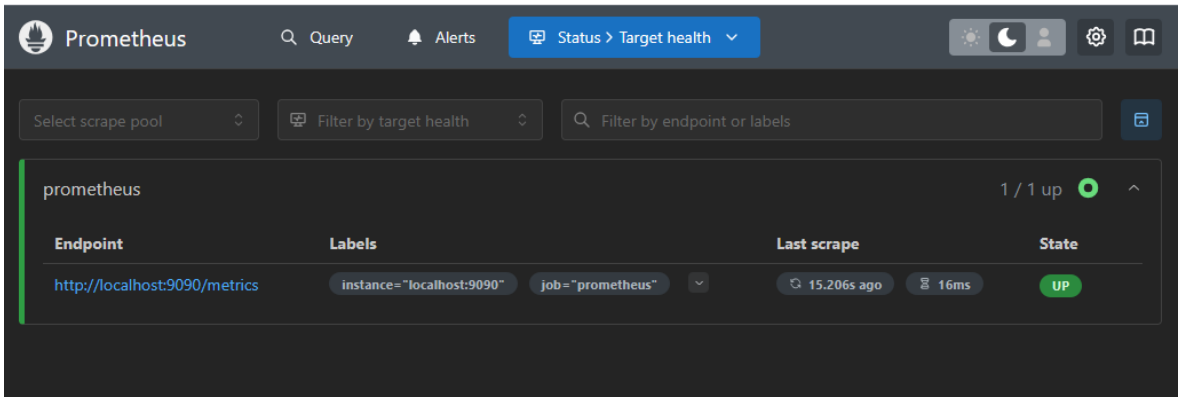
```
ubuntu@k6:~/prometheus-3.1.0.linux-amd64$ sudo systemctl status prometheus
● prometheus.service - Prometheus
     Loaded: loaded (/etc/systemd/system/prometheus.service; enabled; preset: enabled)
     Active: active (running) since Mon 2025-04-28 07:28:15 UTC; 4 days ago
   Main PID: 150842 (prometheus)
      Tasks: 11 (limit: 9258)
     Memory: 92.4M (peak: 150.3M)
        CPU: 44min 386ms
     CGroup: /system.slice/prometheus.service
             └─150842 /usr/local/bin/prometheus --config.file /etc/prometheus/prometheus.yml
```

Kuvio 29. Prometheuksen tila aktiivinen

Lopuksi käydään katsomassa Prometheuksen web käyttöliittymää. Se on hyvä tarkastaa nyt, koska sinne tulee myöhemmin lisää kohteita.



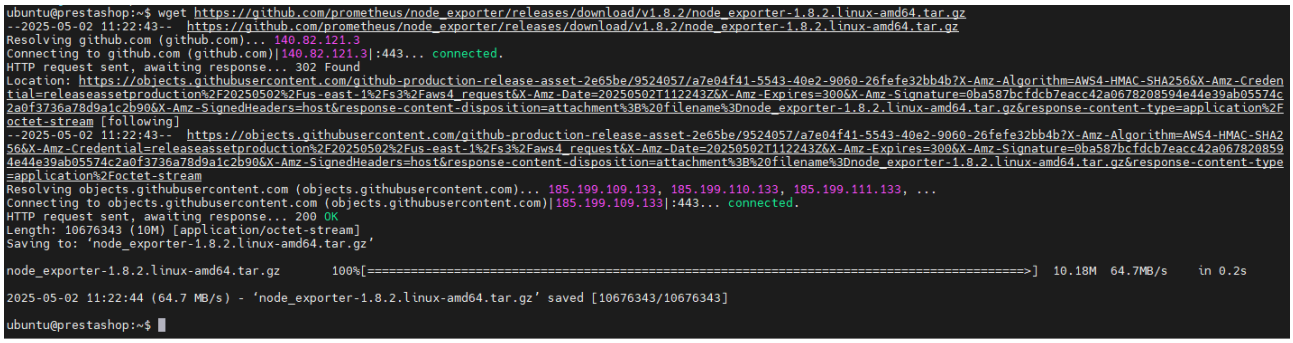
Kuvio 30. Prometheus web käyttöliittymä



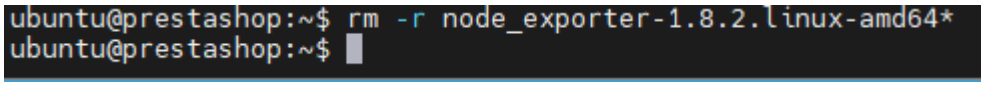
Kuvio 31. Prometheus kohteen tarkastelu

3.2.3 Node Export

Node Exportin asentaminen tapahtuu pitkälti samoin kuten Prometheuksenkin. Aloitetaan lataamalla asennuspaketti ja asennetaan se. Sitten siirretään asennettu sisältö /usr/local/bin, tämän jälkeen poistetaan asennuspaketti.

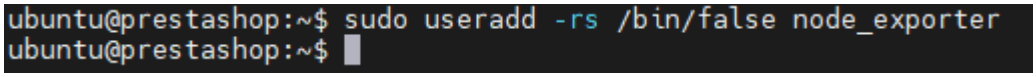


Kuvio 32. Node Exporter asennuspaketin lataus

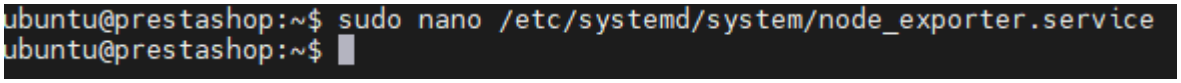


Kuvio 33. Poistetaan Node Exporter asennuspaketti asennuksen jälkeen

Kun Node Exporter on asennettu, siirretty ja asennuspaketti poistettu, lisätään Node Exporterille käyttäjä sekä systemd.service tiedosto.



Kuvio 34. Node Exporter käyttäjän luonti



Kuvio 35. Node Exporter service tiedoston luonti

```
[Unit]
Description=Node Exporter
Wants=network-online.target
After=network-online.target

[Service]
User=node_exporter
Group=node_exporter
Type=simple
ExecStart=/usr/local/bin/node_exporter

[Install]
WantedBy=default.target
```

Kuvio 36. Node Exporter service tiedoston sisältö

Kun node_exporter.service tiedosto on saatu valmiiksi, enabloidaan Node Exporter ja tarkastetaan sen tila systemctl status komennolla.

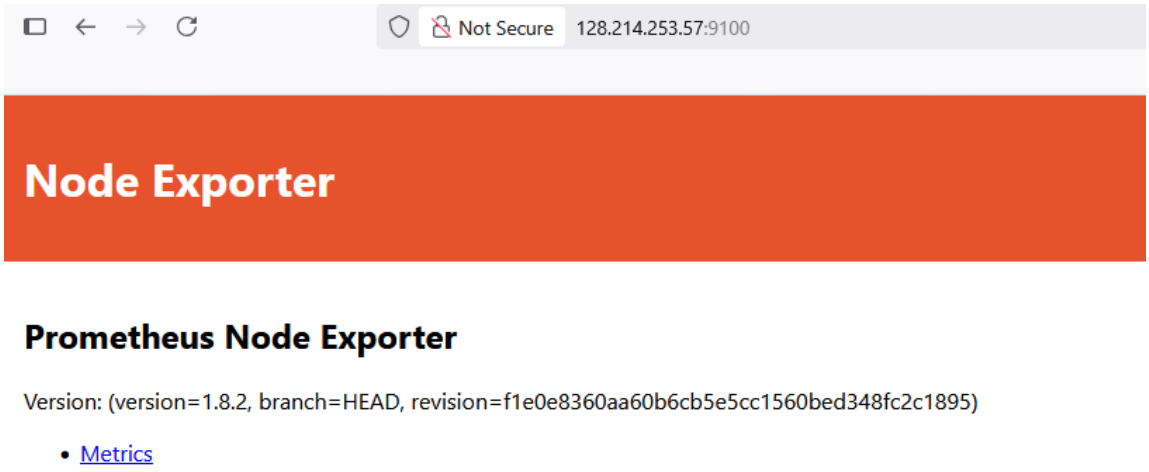
```
ubuntu@prestashop:~$ sudo systemctl enable node_exporter
sudo systemctl daemon-reload
sudo systemctl start node_exporter
Created symlink /etc/systemd/system/default.target.wants/node_exporter.service → /etc/systemd/system/node_exporter.service.
ubuntu@prestashop:~$
```

Kuvio 37. Node Exporter enabloitu

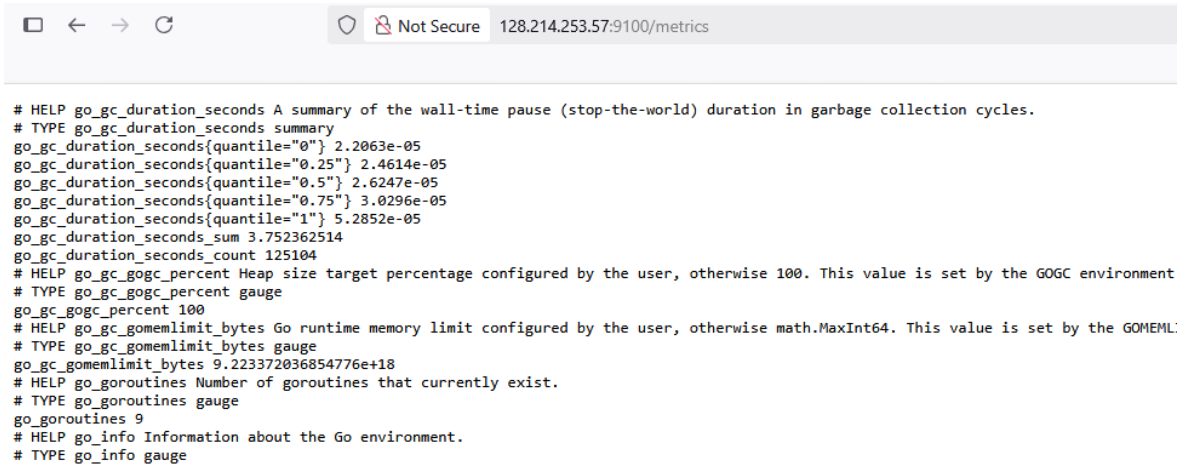
```
ubuntu@prestashop:~$ sudo systemctl status node_exporter
● node_exporter.service - Node Exporter
     Loaded: loaded (/etc/systemd/system/node_exporter.service; enabled; preset: enabled)
     Active: active (running) since Fri 2025-05-02 11:56:31 UTC; 5s ago
   Main PID: 35598 (node_exporter)
      Tasks: 5 (limit: 9258)
     Memory: 2.8M (peak: 2.9M)
        CPU: 6ms
     CGroup: /system.slice/node_exporter.service
             └─35598 /usr/local/bin/node_exporter

May 02 11:56:31 prestashop node_exporter[35598]: ts=2025-05-02T11:56:31.824Z caller=node_exporter.go:118 level=info collector=time
May 02 11:56:31 prestashop node_exporter[35598]: ts=2025-05-02T11:56:31.824Z caller=node_exporter.go:118 level=info collector=timex
May 02 11:56:31 prestashop node_exporter[35598]: ts=2025-05-02T11:56:31.824Z caller=node_exporter.go:118 level=info collector=udp_queues
May 02 11:56:31 prestashop node_exporter[35598]: ts=2025-05-02T11:56:31.824Z caller=node_exporter.go:118 level=info collector=uname
May 02 11:56:31 prestashop node_exporter[35598]: ts=2025-05-02T11:56:31.824Z caller=node_exporter.go:118 level=info collector=vmstat
May 02 11:56:31 prestashop node_exporter[35598]: ts=2025-05-02T11:56:31.824Z caller=node_exporter.go:118 level=info collector=watchdog
May 02 11:56:31 prestashop node_exporter[35598]: ts=2025-05-02T11:56:31.824Z caller=node_exporter.go:118 level=info collector=xfs
May 02 11:56:31 prestashop node_exporter[35598]: ts=2025-05-02T11:56:31.824Z caller=node_exporter.go:118 level=info collector=zfs
May 02 11:56:31 prestashop node_exporter[35598]: ts=2025-05-02T11:56:31.824Z caller=tls_config.go:313 level=info msg="Listening on" address=[::]:9100
May 02 11:56:31 prestashop node_exporter[35598]: ts=2025-05-02T11:56:31.824Z caller=tls_config.go:316 level=info msg="TLS is disabled." http2=false address=[::]:9100
ubuntu@prestashop:~$
```

Kuvio 38. Node Exporter status aktiivinen

Onnistuneen asennuksen ja aktiivi statuksen jälkeen katsotaan miltä Node Exporter näyttää ja näkyykö sen muuntamat metriikat.



Kuvio 39. Node Exporter etusivu



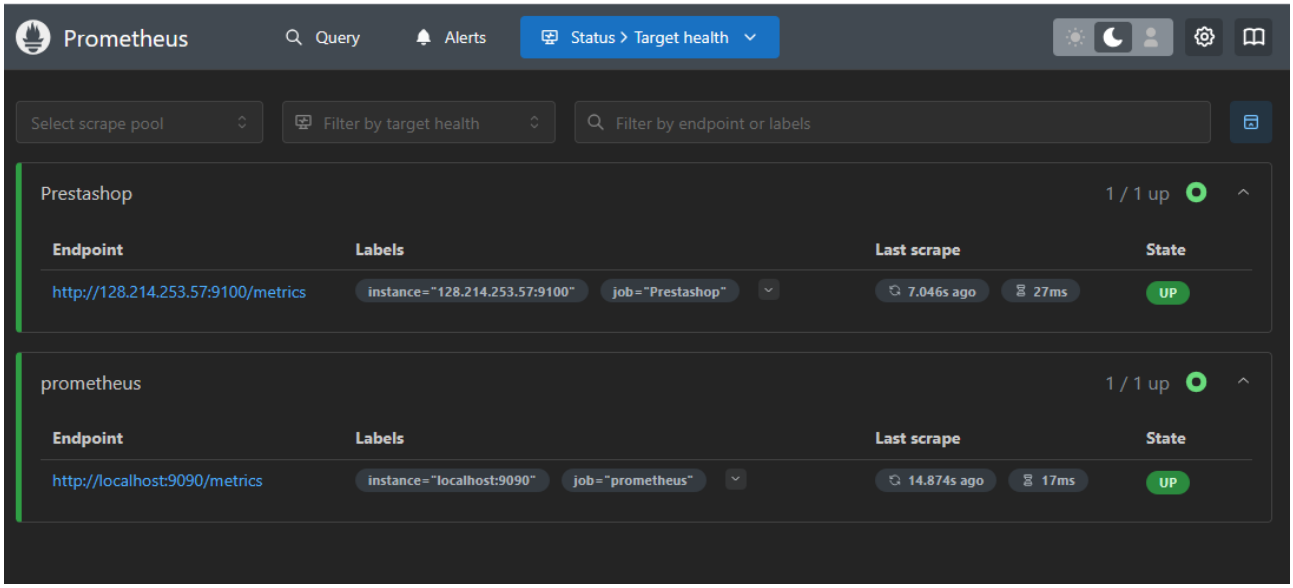
Kuvio 40. Node Exporter metriikoita

Metriikkojen tarkastamisen jälkeen muutetaan K6 koneella /etc/prometheus/prometheus.yml tiedostoa, jotta Prometheus löytäisi Node Exporterin antamat metriikat. Koska oletuksena metriikat löytyvät /metrics kohdasta, lisätään työksi Node Exporter osoite, joka tarkastettiin aikaisemmin.

```
  GNU nano 7.2                                  /etc/prometheus/prometheus.yml *
# my global config
global:
  scrape_interval: 15s # Set the scrape interval to every 15 seconds. Default is every 1 minute.
  evaluation_interval: 15s # Evaluate rules every 15 seconds. The default is every 1 minute.
  # scrape_timeout is set to the global default (10s).

# Alertmanager configuration
alerting:
  alertmanagers:
    - static_configs:
        - targets:
          # - alertmanager:9093

# Load rules once and periodically evaluate them according to the global 'evaluation_interval'.
rule_files:
  # - "first_rules.yml"
  # - "second_rules.yml"

# A scrape configuration containing exactly one endpoint to scrape:
# Here it's Prometheus itself.
scrape_configs:
  # The job name is added as a label `job=<job_name>` to any timeseries scraped from this config.
  - job_name: "prometheus"

    # metrics_path defaults to '/metrics'
    # scheme defaults to 'http'.

    static_configs:
      - targets: ["localhost:9090"]

  - job_name: "Prestashop"
    static_configs:
      - targets: ["128.214.253.57:9100"]
```

Kuvio 41. Node Exporterin lisäys prometheus.yml tiedostoon

Yaml tiedoston muutosten jälkeen käynnistetään Prometheus uudestaan systemctl restart komennolla. Nyt Prometheus sai yhteyden Node Exportin antamiin metriikoihin, ja Prestashop virtuaalikone ilmesty Prometheuksen kohteisiin.



Kuvio 42. Prestashop virtuaalikone prometheus kojelaudan kohteissa

3.2.4 Grafana

Ennen itse Grafanan pakettivaraston lataamista K6 virtuaalikoneelle, tulee ladata Grafanan vaatimat riippuvuudet. Myös GPG-allekirjoitusavain on ladattava ja tallennettava, jotta voidaan varmistaa, että ladattavat paketit ovat aitoja. Grafanan lisäys GPG-allekirjoitusavain lähdeluetteloon, mahdollistaa sen asennuksen apt-get komennolla ja automaatisen päivittymisen muiden järjestelmäpäivitysten mukana.

```
ubuntu@k6:~$ sudo apt-get install -y apt-transport-https software-properties-common
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  python3-software-properties
The following NEW packages will be installed:
  apt-transport-https
The following packages will be upgraded:
  python3-software-properties software-properties-common
2 upgraded, 1 newly installed, 0 to remove and 20 not upgraded.
Need to get 48.2 kB of archives.
After this operation, 36.9 kB of additional disk space will be used.
Get:1 http://nova.clouds.archive.ubuntu.com/ubuntu noble/universe amd64 apt-transport-https all 2.7.14build2 [3974 B]
Get:2 http://nova.clouds.archive.ubuntu.com/ubuntu noble-updates/main amd64 software-properties-common all 0.99.49.2 [14.4 kB]
Get:3 http://nova.clouds.archive.ubuntu.com/ubuntu noble-updates/main amd64 python3-software-properties all 0.99.49.2 [29.8 kB]
Fetched 48.2 kB in 1s (79.4 kB/s)
Selecting previously unselected package apt-transport-https.
(Reading database ... 106291 files and directories currently installed.)
Preparing to unpack .../apt-transport-https_2.7.14build2_all.deb ...
Unpacking apt-transport-https (2.7.14build2) ...
Preparing to unpack .../software-properties-common_0.99.49.2_all.deb ...
Unpacking software-properties-common (0.99.49.2) over (0.99.49.1) ...
Preparing to unpack .../python3-software-properties_0.99.49.2_all.deb ...
Unpacking python3-software-properties (0.99.49.2) over (0.99.49.1) ...
Setting up apt-transport-https (2.7.14build2) ...
Setting up python3-software-properties (0.99.49.2) ...
Setting up software-properties-common (0.99.49.2) ...
Processing triggers for man-db (2.12.0-4build2) ...
Processing triggers for dbus (1.14.10-4ubuntu4.1) ...
Scanning processes...
Scanning candidates...
Scanning linux images...

Pending kernel upgrade!
Running kernel version:
  6.8.0-57-generic
Diagnostics:
  The currently running kernel version is not the expected kernel version 6.8.0-58-generic.

Restarting the system to load the new kernel will not be handled automatically, so you should consider rebooting.

Restarting services...

Service restarts being deferred:
 /etc/needrestart/restart.d/dbus.service
 systemctl restart systemd-logind.service
 systemctl restart unattended-upgrades.service

No containers need to be restarted.

User sessions running outdated binaries:
 ubuntu @ session #1: dirmngr[1424], gpg-agent[1427]
 ubuntu @ user manager service: systemd[1118]

No VM guests are running outdated hypervisor (qemu) binaries on this host.
ubuntu@k6:~$
```

Kuvio 43. Grafana riippuvuuksien lataus

```
ubuntu@k6:~$ sudo wget -q -O /usr/share/keyrings/grafana.key https://apt.grafana.com/gpg.key
ubuntu@k6:~$
```

Kuvio 44. Ladataan ja tallennetaan GPG-allekirjoitusavain

```
ubuntu@k6:~$ echo "deb [signed-by=/usr/share/keyrings/grafana.key] https://apt.grafana.com stable main" | sudo tee -a /etc/apt/sources.list.d/grafana.list
deb [signed-by=/usr/share/keyrings/grafana.key] https://apt.grafana.com stable main
ubuntu@k6:~$
```

Kuvio 45. Lisätään Grafanan pakettivarasta lähdeluetteloon

```
ubuntu@k6:~$ sudo apt-get update
Get:1 https://apt.grafana.com stable InRelease [7660 B]
Get:2 http://security.ubuntu.com/ubuntu noble-security InRelease [126 kB]
Hit:3 https://dl.k6.io/deb stable InRelease
Get:4 http://nova.clouds.archive.ubuntu.com/ubuntu noble InRelease [256 kB]
Get:5 https://apt.grafana.com stable/main amd64 Packages [388 kB]
Get:6 http://security.ubuntu.com/ubuntu noble-security/main amd64 Components [21.5 kB]
Get:7 http://security.ubuntu.com/ubuntu noble-security/universe amd64 Components [52.2 kB]
Get:8 http://security.ubuntu.com/ubuntu noble-security/restricted amd64 Components [208 B]
Get:9 http://security.ubuntu.com/ubuntu noble-security/multiverse amd64 Components [208 B]
Get:10 http://nova.clouds.archive.ubuntu.com/ubuntu noble-updates InRelease [126 kB]
Get:11 http://nova.clouds.archive.ubuntu.com/ubuntu noble-backports InRelease [126 kB]
Fetched 1104 kB in 1s (796 kB/s)
Reading package lists... Done
N: Missing Signed-By in the sources.list(5) entry for 'http://nova.clouds.archive.ubuntu.com/ubuntu'
ubuntu@k6:~$
```

Kuvio 46. Päivitetään pakettilista

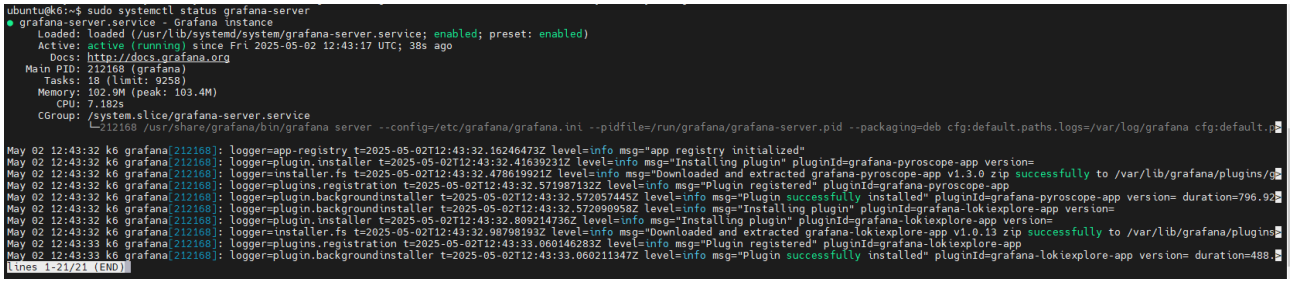
Kun pakettilista on päivitetty, voidaan asentaa Grafana, enabloida se ja varmistaa status komennolla asennuksen ja käynnistyksen onnistuminen.

```
ubuntu@k6:~$ sudo apt-get install grafana -y
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  musl
The following NEW packages will be installed:
  grafana musl
0 upgraded, 2 newly installed, 0 to remove and 20 not upgraded.
Need to get 170 MB of archives.
After this operation, 633 MB of additional disk space will be used.
Get:1 http://nova.clouds.archive.ubuntu.com/ubuntu noble/universe amd64 musl amd64 1.2.4-2 [416 kB]
Get:2 https://apt.grafana.com stable/main amd64 grafana amd64 11.6.1 [170 MB]
Fetched 170 MB in 9s (18.8 MB/s)
Selecting previously unselected package musl:amd64.
(Reading database ... 106295 files and directories currently installed.)
Preparing to unpack .../musl_1.2.4-2_amd64.deb ...
Unpacking musl:amd64 (1.2.4-2) ...
Selecting previously unselected package grafana.
Preparing to unpack .../grafana_11.6.1_amd64.deb ...
Unpacking grafana (11.6.1) ...
Setting up musl:amd64 (1.2.4-2) ...
Setting up grafana (11.6.1) ...
info: Selecting UID from range 100 to 999 ...

info: Adding system user `grafana' (UID 109) ...
info: Adding new user `grafana' (UID 109) with group `grafana' ...
info: Not creating home directory `/usr/share/grafana'.
### NOT starting on installation, please execute the following statements to configure grafana to start automatically using systemd
 sudo /bin/systemctl daemon-reload
 sudo /bin/systemctl enable grafana-server
### You can start grafana-server by executing
 sudo /bin/systemctl start grafana-server
Processing triggers for man-db (2.12.0-4build2) ...
Scanning processes...
Scanning candidates...
Scanning linux images...

Pending kernel upgrade!
Running kernel version:
  6.8.0-57-generic
Diagnostics:
  The currently running kernel version is not the expected kernel version 6.8.0-58-generic.

Restarting the system to load the new kernel will not be handled automatically, so you should consider rebooting.

Restarting services...

Service restarts being deferred:
 /etc/needrestart/restart.d/dbus.service
 systemctl restart systemd-logind.service
 systemctl restart unattended-upgrades.service

No containers need to be restarted.

User sessions running outdated binaries:
 ubuntu @ session #1: dirmngr[1424], gpg-agent[1427]
 ubuntu @ user manager service: systemd[1118]

No VM guests are running outdated hypervisor (qemu) binaries on this host.
ubuntu@k6:~$
```

Kuvio 47. Grafanan Asennus

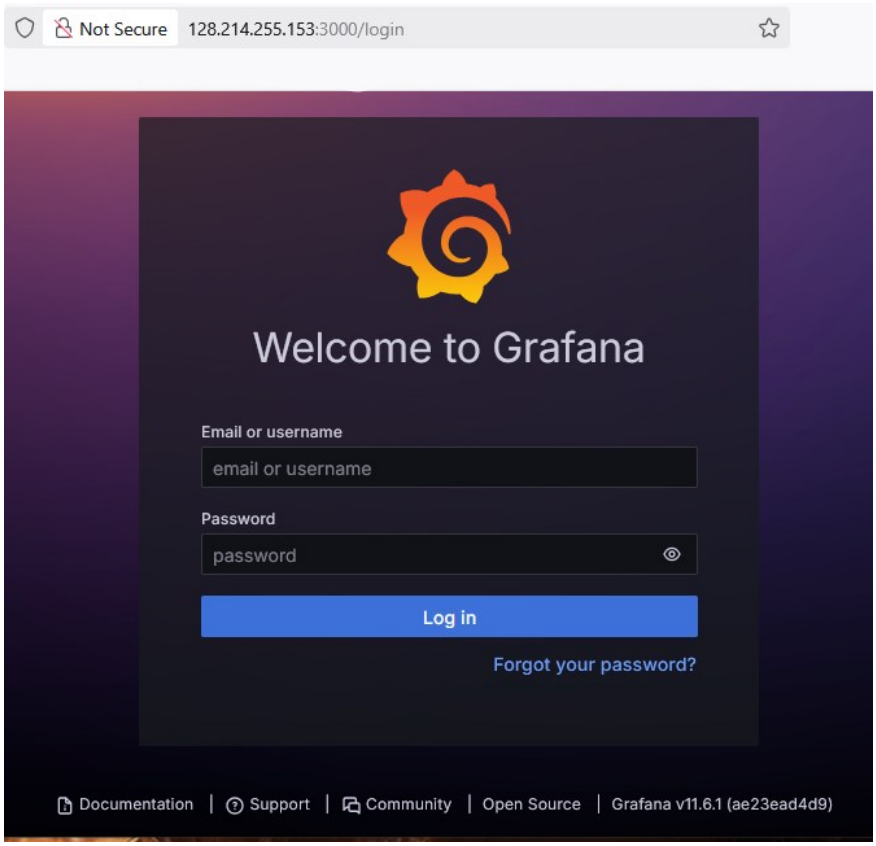
```
ubuntu@k6:~$ sudo systemctl daemon-reload
sudo systemctl enable grafana-server.service
sudo systemctl start grafana-server
Synchronizing state of grafana-server.service with SysV service script with /usr/lib/systemd/systemd-sysv-install.
Executing: /usr/lib/systemd/systemd-sysv-install enable grafana-server
Created symlink /etc/systemd/system/multi-user.target.wants/grafana-server.service → /usr/lib/systemd/system/grafana-server.service.
ubuntu@k6:~$
```

Kuvio 48. Grafanan enablointi



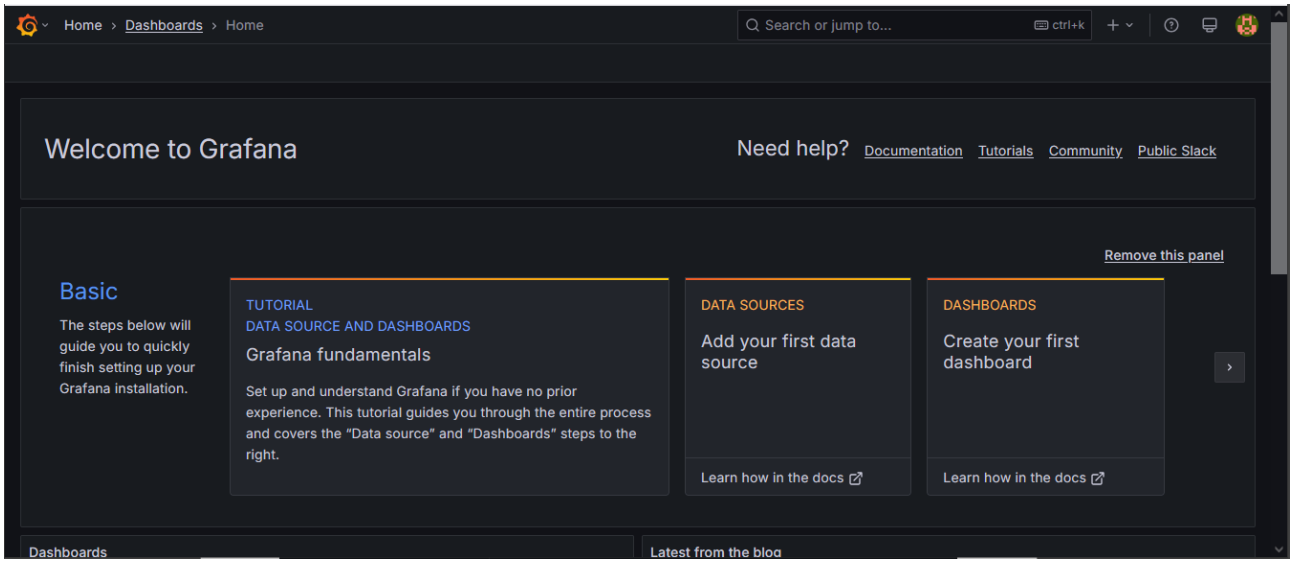
Kuvio 49. Grafanan tilan tarkastaminen

Kun Grafanan tila on aktiivinen, voidaan se avata selaimessa.

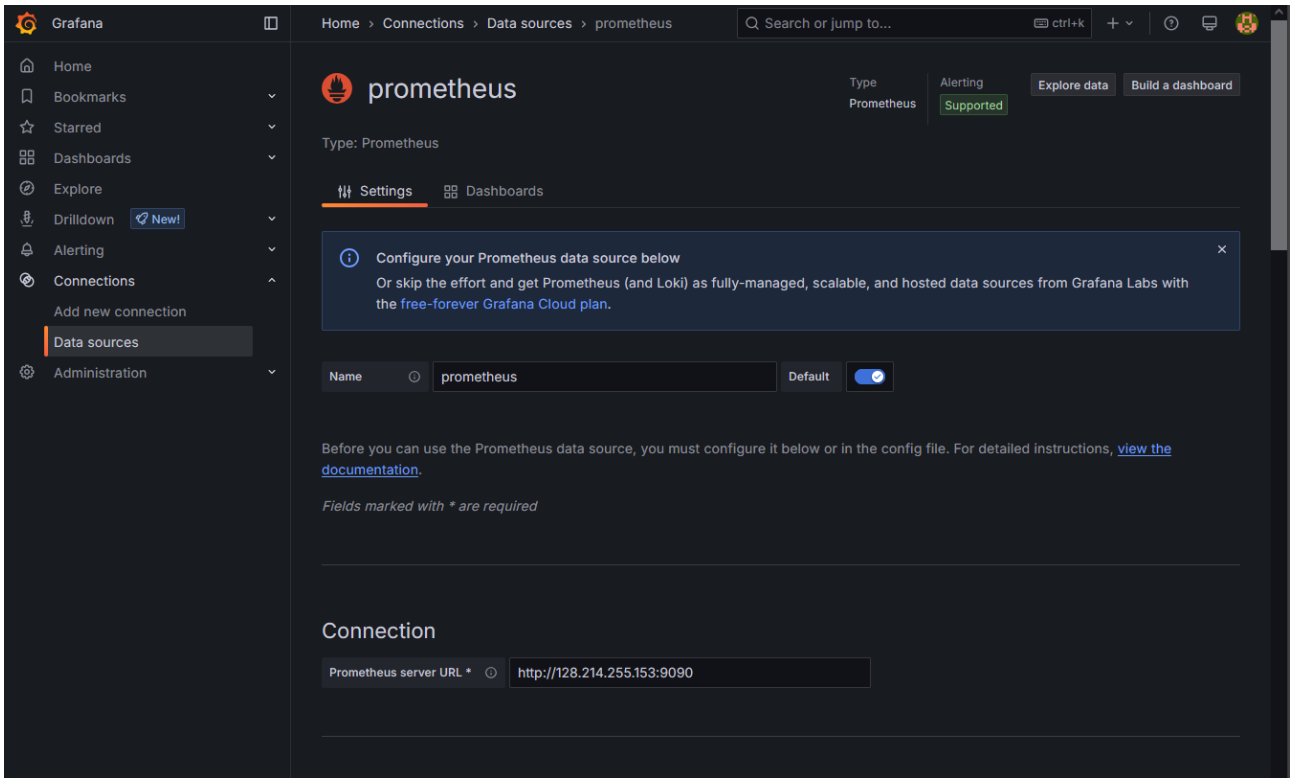


Kuvio 50. Grafana selaimessa

Grafanaan kirjautuessa (Admin / Admin) pyydetään vaihtamaan salasana. Kun Grafanaan on päästy sisälle, tulee sille visualisointia varten laittaa datalähde. Tässä tapauksessa datalähde on aikaisemmin asennettu Prometheus, jotta voimme visualisoida mitä Prestashop VM:ssä tapahtuu.

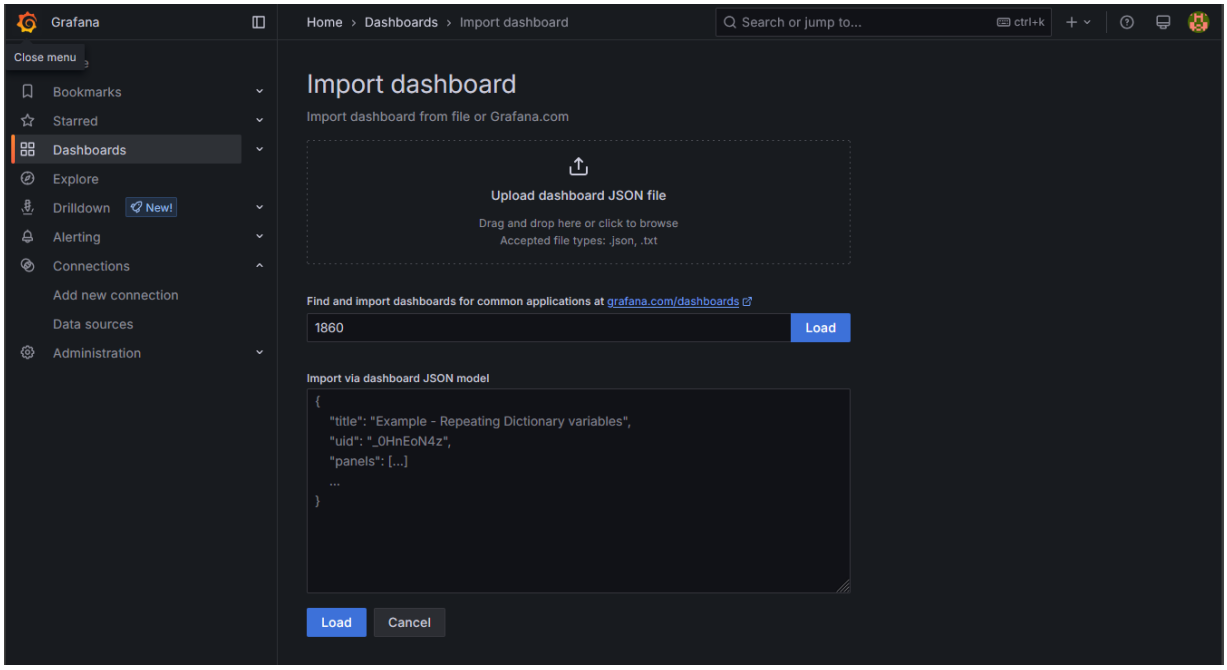


Kuvio 51. Grafana sisäänkirjautumisen jälkeen

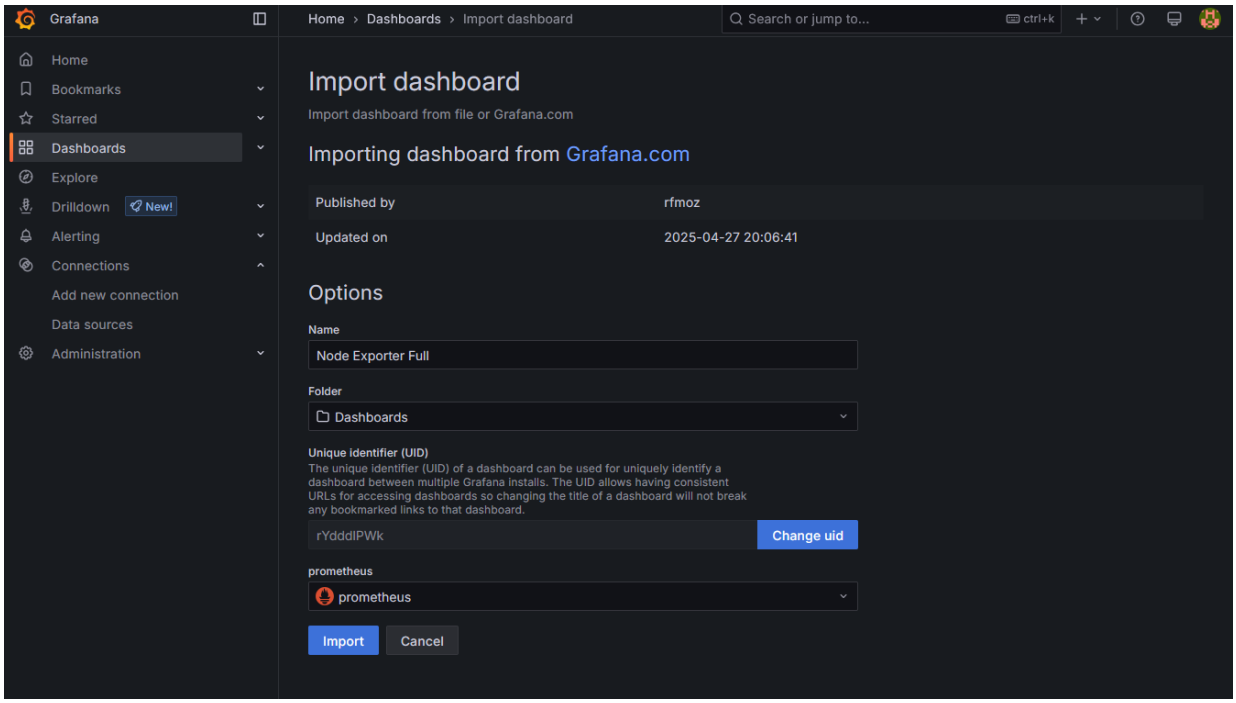


Kuvio 52. Grafana yhdistetään Prometheukseen

Jotta Prometheuksen keräämä data olisi helpommin luettavissa, käytämme kojelautaa sen visualisoimiseksi. Grafanassa kojelautoja on useampia, mutta tässä työssä käytetään rfmoz julkaisemaa Node Exporter Full kojelautaa.



Kuvio 53. Granan kojelaudan etsiminen



Kuvio 54. Haluttu kojelauta

Kuvio 55. Kojelauta ulkonäkö ilman käynnissä olevia testejä

3.3 K6 Suorituskykytestaus

Suorituskykytestausta aloitettaessa tuli päättää mitä testejä tultiin käyttämään. Paras käytäntö oli tietenkin kokeilla kaikkea, jotta saatiin laajin mahdollinen kuva suorituskyvystä. Aina kaiken testaaminen ei ole mahdollista, jolloin piti valita haluttuun tarkoitukseen soveltuvat testit.

Kuormitustestaus oli hyvä, yleinen testi, jolla simuloitiin sovellukselle odotettua keskivertoliikennettä. Seuraamalla keskivertoliikenteen vaikutuksen seuraamisella voitiin saada selville jokapäiväisen käyttäjä kokemus suorituskyvyn näkökulmasta.

Alla olevassa kuviossa on JavaScript-ohjelmointikielellä tehty kuormitustestaus testi. Alussa tuotiin tarvitut kirjastot, tässä tapauksessa tarvittiin vain k6/http ja k6 kirjastot. Sen jälkeen testille asetettiin vaiheet. Tehdyssä testissä oli kolme vaihetta nousu, pito ja lasku. Näissä vaiheissa annettiin haluttu kesto ja virtuaalikäyttäjien (VUs) määrä.

Ensimmäisessä vaiheessa, eli nousussa, annettiin kestoksi kolme minuuttia ja virtuaalikäyttäjäkohteeksi 100. Tämä tarkoitti sitä, että kolmen minuutin aikana virtuaalikäyttäjien määrä nousee nollasta sataan. Toisessa vaiheessa, eli pidossa, kestoksi asetettiin 30 minuuttia ja virtuaalikäyttäjien määräksi sama kuin vaiheessa yksi, eli 100. Koska ensimmäisessä ja toisessa vaiheessa VU määrä

on sama, K6 tulkitsi sen tarkoittavan, että ensimmäisessä vaiheessa nostetaan VU määrä ja toisessa vaiheessa sitä pidetään määrätty aika. Kolmannessa vaiheessa, eli laskussa, kestoksi annettiin kolme minuuttia ja virtuaalikäyttäjien määräksi nolla. Tämä tarkoitti sitä, että VU määrä pudotetaan nollaan kolmen minuutin aikana. Koska kolmas vaihe oli viimeinen ja VU kohde nolla, testi tuli loppumaan, kun haluttu virtuaalikäyttäjä määrä saavutettiin.

Vaiheiden jälkeen testille asetettiin haluttuja kynnyksiä. Testissä haluttiin seurata virtuaalikäyttäjien http-pyyntöjen vastausnopeutta sekä http-pyyntöjen epäonnistumisprosenttia. Vastausnopeus asetettiin niin, että 95 prosenttia http-pyynnöistä tulee valmistua nopeammin kuin 500 millisekuntia. Testin epäonnistumisprosentti tulisi olla alle 1. Nopea http-pyyntö valmistumisaika ja pieni epäonnistumisprosentti antavat käyttäjälle sulavan kokemuksen.

Lopuksi testiin laitettiin funktio, jossa kerrottiin testin kohde sekä laitettiin ehdot, jotka tarkistavat, että http-vastauksen statuskoodi on 200, eli pyyntö onnistui ja vasteaika oli alle 500 millisekuntia. Viimeisenä testissä oli sleep, joka antoi nollan ja kahden sekunnin välillä olevan satunnaisen viiveen, joka simuloi realistisempaa liikettä. Ilman sleep:iä kaikki pyynnöt lähtisivät samaan aikaan.

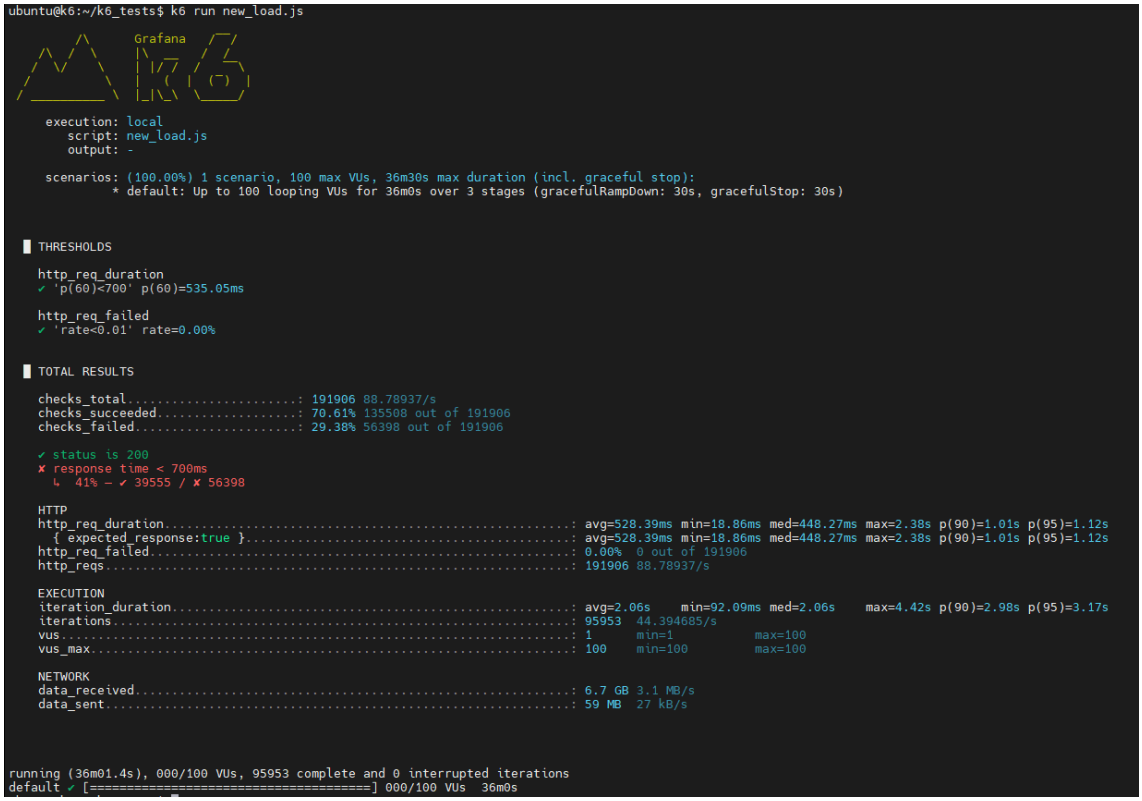
```
  GNU nano 7.2                                                    load.js
import http from 'k6/http';
import { check, sleep  } from 'k6';

export let options = {
  stages:[
    { duration: '3m', target: 100 }, // Ramp-up to 100 virtual users in 3 minutes
    { duration: '30m', target: 100 }, // Hold 100 VUs for 30 minutes
    { duration: '3m', target: 0 }, // Ramp-down to 0 virtual users over 3 minutes


  ],
  thresholds: {
    http_req_duration: ['p(95)<500'], // 95% of requests should be < 500ms
    http_req_failed: ['rate<0.01'],   // Error rate should be less than 1%
  },
};

export default function () {
  const res = http.get('https://vm3856.kaj.pouta.csc.fi');
  check(res, {
    'status is 200': (r)=> r.status === 200,
    'response time < 500ms': (r) => r.timings.duration < 500,
  });
   sleep(Math.random() * 2); // Sleep between 0 and 2 seconds
}
```

Kuvio 56. load.js kuormitustestaus testi

Kuvio 57. Grafana load.js testi loppunut

```
ubuntu@k6:~/k6_tests$ k6 run load.js

        /\      Grafana   /‾‾/
   /\  /  \     |\  __   /  /
  /  \/    \    | |/ /  /   ‾‾\
 /          \   |   (  |  (‾)  |
/ __________ \  |_|\_\  \_____/

    execution: local
       script: load.js
       output: -

    scenarios: (100.00%) 1 scenario, 100 max VUs, 36m30s max duration (incl. graceful stop):
             * default: Up to 100 looping VUs for 36m0s over 3 stages (gracefulRampDown: 30s, gracefulStop: 30s)


  THRESHOLDS

    http_req_duration
    ✗ 'p(95)<500' p(95)=931.69ms

    http_req_failed
    ✓ 'rate<0.01' rate=0.00%


  TOTAL RESULTS

    checks_total.....................: 211188 97.726206/s
    checks_succeeded...................: 66.01% 139407 out of 211188
    checks_failed.....................: 33.98% 71781 out of 211188

    ✓ status is 200
    ✗ response time < 500ms
      ↳  32% — ✓ 33813 / ✗ 71781

    HTTP
    http_req_duration.....................................................: avg=432.82ms min=17.12ms med=364.16ms max=2.46s p(90)=833.07ms p(95)=931.69ms
      { expected_response:true }...........................................: avg=432.82ms min=17.12ms med=364.16ms max=2.46s p(90)=833.07ms p(95)=931.69ms
    http_req_failed.....................................................: 0.00%  0 out of 211188
    http_reqs.....................................................: 211188 97.726206/s

    EXECUTION
    iteration_duration.....................................................: avg=1.87s    min=77.41ms med=1.87s    max=4.27s p(90)=2.75s    p(95)=2.93s
    iterations.....................................................: 105594 48.863103/s
    vus.....................................................: 1      min=1          max=100
    vus_max.....................................................: 100    min=100        max=100

    NETWORK
    data_received.....................................................: 7.3 GB 3.4 MB/s
    data_sent.....................................................: 65 MB  30 kB/s



running (36m01.0s), 000/100 VUs, 105594 complete and 0 interrupted iterations
default ✓ [======================================] 000/100 VUs  36m0s
ERRO[2161] thresholds on metrics 'http_req_duration' have been crossed
ubuntu@k6:~/k6_tests$
```

Kuvio 58. K6 load.js testin tulos, p (95) <500 ms



Kuvio 59. K6 new_load.js testin tulos, p (60) <700 ms

Load.js testin suorituksen jälkeen K6 näytti tietoa tehdystä testistä. (Ks. kuvio 58.) Niitä tietoja tarkastellen voitiin arvioida kohteen suorituskykyä. Tässä testissä, joka kesti 36 minuuttia, ehdittiin tehdä 105594 testikierrosta (http-pyyntö ja vastaus). Siinä näkyi myös asetettujen kynnysten tulos, tässä tapauksessa virheitten määrä oli alle prosentin, mutta vain 32 % (33813) vastauksista alitti 500 millisekunnin rajan, 68 % vastauksista oli alle 500 ms. Oli myös muistettava, että tulokset saattavat heittää muutaman prosentin useista eri syistä riippuen, sen takia oli tärkeää tehdä paljon testausta eri sisällöillä.

Valmiista testistä tutkittiin neljää osaa eri osaa, prosenttipiste, aikaraja, läpäisyprosentti ja vasteaika. Prosenttipiste toimi keskiarvon tilalla, sillä keskiarvossa yksi hidas vastaus vääristi koko tuloksen. Jos prosenttipiste oli p (95), niin 95 prosenttia vasteajoista on nopeampia, kun annettu prosenttimäärä. Vasteaika toimi käsi kädessä prosenttipisteen kanssa. Esimerkiksi p (95) <500 vasteaika oli 931.69 millisekuntia, eli 95 % vasteajoista oli nopeampia kuin 931.69 millisekuntia ja 5 % oli hitaampia. (Ks. Kuvio 58, jossa näkyy esimerkki testin tulokset)

Aikaraja on ennalta annettu tavoite vasteajalle. Sen avulla voidaan arvioida pyyntöjen nopeutta ja sitä voidaan pitää vertailukohteena. Aikaisemmassa esimerkissä käytettiin p (95) <500. Siinä asetettu vasteaika on 500 millisekuntia eli 95 % pyynnöistä tulisi olla alle 500 ms. Läpäisyprosentti kertoi, kuinka suuri osa pyynnöistä läpäisi annetun aikarajan. Samassa esimerkissä, taulukossa 1 katsoen, p (95) <500 läpipääsyprosentti oli 32 %. Tämä tarkoitti sitä, että kaikista testin aikana tehdyistä pyynnöistä, vain 32 % oli nopeampia kuin 500 millisekuntia.

Taulukko 1. K6 kuormitustestien tuloksia.

Prosentti-piste	Aikaraja (ms)	Läpäisy %	Epäonnistu-mis %	Läpäisseet pyynnöt	Epäonnistuneet pyynnöt	Vasteaika (ms)
p (95)	<900	56	44	51703	39934	1200
p (95)	<700	41	59	39913	56320	1110
p (95)	<500	32	68	33813	71781	931.69
p (95)	<300	9	91	8499	83356	1200
p (60)	<900	62	38	59044	35270	549.93
p (60)	<700	41	59	39555	56398	535.05
p (60)	<500	21	79	20422	75244	539.4
p (60)	<300	9	91	8446	83105	585.76
p (40)	<900	64	36	61723	33323	390.73

p (40)	<700	40	60	38405	57068	387.95
p (40)	<500	22	78	21299	75393	379.8
p (40)	<300	9	91	8647	84570	405.91

4 Tulokset ja pohdinta

k6-mcp ja AI testauksen tulevaisuus? kirjoita siitä jotakin omaan **Tulevaisuus** osioon.

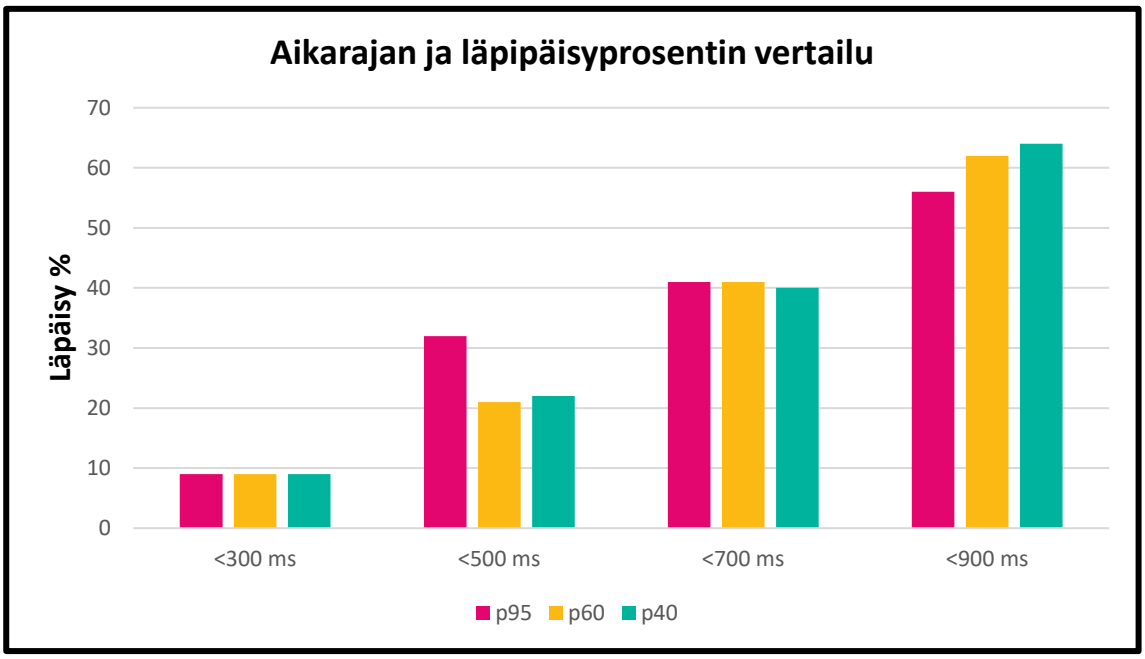
Tehdyissä suorituskykytesteissä oli tarkasteltu prosenttipisteitä, aikarajoja, läpäisyprosentteja sekä vasteaikoja. Tulokset olivat kerätty K6-suorituskykytestaus työkalun avulla eri kuormitustilanteissa. Taulukosta näkyi kuinka, prosenttipiste ja aikaraja vaikuttivat läpipääsy % ja vasteaikaan. (Ks. Taulukko 1).

Suoritetuista kuormitustestaus testeistä voidaan huomata, kuinka järjestelmän vasteajat vaihtelevat kuormituksen ja asetettujen ehtojen mukaan. Oli havaittavissa, että viisi kahdestatoista testistä onnistui pitämään vasteajan aikarajan alapuolella, kun taas seitsemän testin vasteajat menivät aikarajan ylitse. Esimerkiksi p (40) <900 ms suoriutui parhaiten kaikista testeistä 64 % läpäisyprosentilla, kun taas huonoiten pärjäsi p (95) <300 vain 9 % läpipääsyprosentilla ja 1200 millisekunnin vasteajalla.

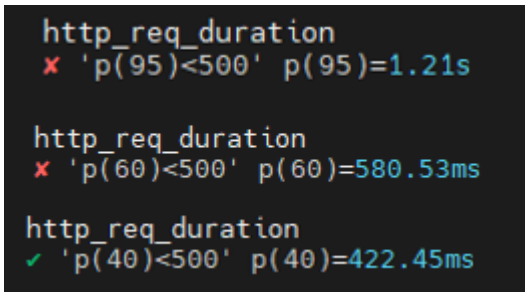
Vasteaikojen perusteella järjestelmä osoitti pystyvänsä käsittelemään kevyttä ja keskiraskasta kuormaa hyväksyttävästi. Kuorman kasvaessa järjestelmän suorituskyky heikkeni nopeasti, eikä pystynyt toimimaan vaaditulla tasolla. Kuormitustestien tuloksia tutkiessa, huomattiin seuraavanlaiset havainnot, vasteaika korreloituu prosenttipisteiden kanssa ja aikaraja korreloituu läpäisy prosenttien kanssa.

Vasteaika ja prosenttipisteet ovat suhteessa toisiinsa niin, että mitä suurempi prosenttipiste oli käytössä, sitä suurempi vasteaika. Syy tähän oli, koska suurempi prosenttipiste kuvasi suurempia

osuuksia vasteajoita, sisältäen enemmän hitaampia pyyntöjä. Pienillä prosenttipisteillä vasteajat olivat nopeita, koska ne edustivat pienempää osuutta kokonaisuudesta. Aikarajan ja läpäisyprosentin välinen korrelaatio toimivat niin, että suurempi aikaraja mahdollistaa hitaampien pyyntöjen läpipääsyn. Samalla periaatteella lyhyt aikaraja antaa vain nopeimmille mahdollisuuden päästä läpi.



Kuvio 60. Aikarajan ja läpipäisyprosentin vertailu



Kuvio 61. New_load.js testin vasteaikoja eri prosenttipisteillä

Tulosten perusteella järjestelmä suoriutuu parhaiten kevyessä kuormassa, kun aikarajat ovat sallivampia ja prosenttipisteet matalampia. Suorituskyky heikkenee merkittävästi kuormituksen kasvaessa, mikä viittaa rajalliseen skaalautuvuuteen. Järjestelmän parantamiseksi tulisi harkita resurssien lisäämistä tai suorituskyvyn optimointia.

Opinnäytetyön alussa suurin osa ajasta kului uuden oppimiseen ja työympäristöihin tutustumiseen. En ollut koskaan aiemmin käyttänyt tai pystyttänyt verkkokauppaa, joten työ oli aluksi haastavaa. Onneksi tehtävää ei tarvinnut suorittaa yksin, sillä koko Task Force -tiimi oli mukana, ja tukea oli hyvin saatavilla.

Alkuvaikeuksien jälkeen työ alkoi sujua, kun jokainen löysi oman roolinsa ja erikoistui omaan osa-alueeseensa. Vaikka pääsin keskittymään minua kiinnostavaan aiheeseen, työ pysyi koko ajan vaativana. Suurin haaste koko opinnäytetyön aikana oli aika – asioita, joihin olisi halunnut perehtyä tarkemmin, oli paljon enemmän kuin aavistinkaan.

Erityisesti Model Context Protocol (MCP) herätti mielenkiintoni, mutta kuulin siitä valitettavasti vasta työn loppupuolella. MCP:n tarjoamat mahdollisuudet testiautomaation saralla ovat merkittäviä, erityisesti kyky luoda testejä ilman manuaalisia syötteitä ja suorittaa API-testejä ilman koodia.

Testauksen tulevaisuus näyttää selvästi suuntautuvan kohti tekoälyn hyödyntämistä. Vaikka tekoäly avaa uusia ovia ja helpottaa monia prosesseja, on syytä heijastaa sitä tuleen, joka on hyvä renki, mutta huono isäntä.

Lähteet

Bigelow, J.S. & Gillis, S.A. 2024. What is multi-tenancy (multi-tenant architecture)? Kirjoitus TechTarget sivustolla. Viitattu 6.4.2025. https://www.techtarget.com/whatis/definition/multi-tenancy

eCommerce Usage Distribution in the Top 1 Million Sites. N.d. Dataa verkkokauppa-alustojen suosiosta. Viitattu 16.5.2025. https://trends.builtwith.com/shop

Extensions. N.d. Grafanal Labs. Grafana k6 dokumentaatio. Viitattu 5.3.2025. https://grafana.com/docs/k6/latest/extensions/

Flinders, M. & Smalley, I. 2024. What is high availability? Kirjoitus IBM sivustolla. Viitattu 6.4.2025. https://www.ibm.com/think/topics/high-availability

Grafana OSS and Enterprise. N.d. Grafanan virallinen dokumentaatio. Viitattu 6.4.2025. https://grafana.com/docs/grafana/latest/

Harsh. 2024. Prometheus and Service Discovery: A Perfet Match for Dynamic Environments. Viitattu 5.4.2025. https://harsh05.medium.com/prometheus-and-service-discovery-a-perfect-match-for-dynamic-environments-c6b3fe0be5b1

Hooda, I. & Chhillar, R.S. 2015. Software Test Process, Testing Types and Techniques. International Journal of Computer Applications. Volume 111. Number 13. February 2015. Viitattu 2.3.2025. https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=0fbe1b5515e747025d950658fbc039e98b29b801

Hussain, S. (2024). Prometheus Monitoring: Features, Components, Architecture, and Metrics. Blogi postaus K21Academy sivustolla. Viitattu 4.42025. https://k21academy.com/prometheus/prometheus-monitoring-an-introduction/

IT HISTORY SOCIETY. N.d. Load Impact. ithistory nettisivut. Viitattu 5.3.2025. https://do.ithistory.org/db/companies/load-impact?utm_source=chatgpt.com

Jyväskylän ammattikorkeakoulu. N.d. Tietoa Jamkista. Jyväskylän ammattikorkeakoulun nettisivut. Viitattu 7.2.2025. https://www.jamk.fi/fi/jamk/tietoa-jamkista

Kataja, J. 2016. PrestaShop verkkokauppa-alusta. Kirjoitus PrestaShop verkkokauppa-alustasta. Viitattu 16.5.2025. https://www.zoner.fi/verkkokaupan-perustaminen/prestashop/

Kumar, R. 2021. What is Prometheus and How it Works? Blogi DevOpsSchool sivustolla. Viitattu 4.42025. https://www.devopsschool.com/blog/what-is-prometheus-and-how-it-works/

Kyle Wiggers. 2021. Grafana Labs acquires load-testing startup K6. VentureBeat artikkeli. Viitattu 5.3.2025. https://venturebeat.com/business/grafana-labs-acquires-load-testing-startup-k6/

Load Testing – Software Testing. 2024.. Artikkeli geeksforgeeks nettisivulla. Viitattu 3.3.2025. https://www.geeksforgeeks.org/software-testing-load-testing/#what-is-load-testing

Mikä CSC. N.d. Tietopaketti CSC sivuilla. Viitattu 16.4.2025. https://csc.fi/tietoa-meista/mika-csc/

Mit Thakkar. N.d. Importance of Performance Testing in your Development Cycle. Blogi. Viitattu 28.2.2025. https://www.kiwiqa.com/importance-of-performance-testing/

Prometheus Node Exporter Setup. N.d. Ohje Couchbase sivustolla node exporterin pystyttämiseen. Viitattu 6.4.2025. https://developer.couchbase.com/tutorial-node-exporter-setup

Slingerland, C. 2024. Horizontal Vs. Vertical Scaling: Which Should You Choose? Blogi postaus CloudZero nettisivulla. Viitattu 6.4.2025. https://www.cloudzero.com/blog/horizontal-vs-vertical-scaling/

This is k6. N.d. k6 ohjelmiston matka k6 nettisuvuilla. Viitattu 5.3.0205. https://k6.io/about/?utm_source=chatgpt.com

Toikko, T & Rantanen, T. 2009. Tutkimuksellinen kehittämistoiminta. Tampere: Tampereen yliopistopaino. Viitattu 3.2.2025 https://trepo.tuni.fi/handle/10024/100802

Tricentia Staff. 2024. Performance testing, best practices, metrics & more. Artikkeli. Viitattu 19.1.2025. https://www.tricentis.com/learn/performance-testing

Types of load testing. N.d. Grafanan nettisivut. Viitattu 10.3.2025. https://grafana.com/load-testing/types-of-load-testing/

What is Prometheus. N.d. Prometheuksen virallinen dokumentaatio. Viitattu 5.4.2025. https://prometheus.io/docs/introduction/overview/

What is Stress Testing in Software Testing? 2024. Artikkeli geeksforgeeks sivustolla. Viitattu 10.3.2025. https://www.geeksforgeeks.org/stress-testing-software-testing/#what-is-stress-testing

WIMMA Capstone™. N.d. WIMMA Capstonen labranet sivut. Viitattu 8.2.2025. https://wimma-capstone.pages.labranet.jamk.fi/dashboard/

ZAPTEST. N.d. Artikkeli ZAPTEST nettisivulla. Viitattu 10.2.2025. https://www.zaptest.com/fi/mita-on-suorituskykytestaaminen-syvasukellus-tyyppeihin-kaytantoihin-tyokaluihin-haasteisiin-ja-muuhun