LAB University of Applied Sciences

Keskitetyn monitorointinäkymän kehittäminen

LAB-ammattikorkeakoulu
Insinööri (AMK) Tieto- ja Viestintätekniikka
2025
Mikko Sundberg

Tiivistelmä

Tekijä(t)	Julkaisun laji	Valmistumisaika
Mikko Sundberg	Opinnäytetyö, AMK	2025
	Sivumäärä 38	

Työn nimi

Keskitetyn monitorointinäkymän kehittäminen

Tutkinto ja koulutusala

Insinööri (AMK), tieto- ja viestintätekniikka

Toimeksiantajaorganisaatio (jos opinnäytetyöllä on toimeksiantaja)

Elisa OYJ

Tiivistelmä

Työssä käsitellään keskitetyn palvelimien monitorointijärjestelmän pilotin (engl. Proof of Concept) suunnittelua ja toteutusta Elisa Oyj:n Automation & Tools - tiimille. Toimeksiantajalla oli lähtökohtaisesti mahdollisuus monitoroida järjestelmiään, mutta näkymät olivat pirstaloituneet useisiin eri työkaluihin ja käyttöliittymiin. Työn tavoitteena oli kehittää ratkaisu, jolla järjestelmien metriikkadataa voidaan hakea ja esittää yhtenäisesti siten, että ratkaisu on tarvittaessa siirrettävissä eri palvelinympäristöihin ja joustavasti mukautettavissa eri tietolähteisiin.

Teoriaosuudessa käsitellään työssä käytettyjä teknologioita sekä jatkuvan integroinnin ja jatkuvan käyttöönoton (CI/CD) periaatteita. Osuus auttaa ymmärtämään työssä tehtyjä valintoja, sekä antaa yleistason kuvauksen tyypillisistä työkaluista ja käytännöistä, joita nykyaikaisessa infrastruktuurinhallinnassa usein hyödynnetään.

Järjestelmän suunnittelussa ja toteutuksessa täytyi ottaa huomioon, että monitorointijärjestelmä tulee lopulta toimimaan toimeksiantajan Kubernetes-alustalla, eikä mitään järjestelmän osaa voida asentaa kohdepalvelimille. Ratkaisu oli käyttää itse toteutettua metriikkakerääjää, josta metriikkatiedot voidaan noutaa Prometheuksella. Näkymien toteutukseen ja tiedon visualisointiin käytettiin Grafanaa. Järjestelmän kehitysputki toteutettiin Github Actions -työkaluilla.

Lopputuloksena oli toimiva CI/CD-putki järjestelmän kehitystä varten ja minimaalinen versio järjestelmästä, jossa järjestelmän ydinosat on toteutettu ja data kulkee järjestelmän päästä päähän. Lisäksi tiedonkeruulogiikan perustoiminta toteutettiin ja sen toiminta todennettiin paikallisessa kehitysympäristössä.

Asiasanat

Monitorointi, Grafana, Prometheus, Linux, Kubernetes, Go, Github Actions

Abstract

Author(s)	Type of Publication	Published
Mikko Sundberg	Thesis, UAS	2025
	Number of Pages	
	38	

Title of Publication

Developing a centralized monitoring system

Degree, Field of Study

Engineer (UAS), Information and Communication technology

Organisation of the client (if the thesis work is commissioned by another party)

Elisa OYJ

Abstract

This thesis describes the design and implementation of a proof-of-concept solution for a centralized server monitoring system. The client team had some existing monitoring capabilities, but the visualization was fragmented across multiple tools and interfaces. The objective of the work is to come up with a solution for collecting and visualizining metric data in a way that is scalable to diverse environments and adaptable for different types of metric data in the future.

The theory section covers the technologies used in the project at a general level, as well as the principles of Continuous Integration and Continuous Delivery (CI/CD). Additionally, the section provides context for the choices made during implementation by introducing relevant tools and practices commonly used in modern infrastructure management.

In the system design, it was necessary to account for the requirement that the monitoring solution must ultimately run on the client team's Kubernetes platform, with no component deployed directly on the target servers. The implementation included a metrics collector written in Go, which exposes an HTTP interface compatible with the Prometheus standard, because Prometheus was used for data scraping. Grafana was used for creating dashboards and visualizing the data collected. The system's development pipeline was implemented using GitHub Actions.

The end result was a functional CI/CD pipeline to support system development and a minimal working version of the system with end-to-end data flow. The core data collection logic was also implemented and verified to work within a local development environment.

Keywords

Monitoring, Grafana, Prometheus, Linux, Kubernetes, Go, Github Actions

Sisällys

1 Johdanto 1
2 CI/CD: Jatkuva integrointi ja toimitus 2
2.1 CI/CD:n perusteet 2
2.2 Jatkuva integrointi 2
2.3 Jatkuva toimitus ja käyttöönotto 2
3 Linux – komentorivityökalut 4
3.1 Proc-tiedostojärjestelmä 4
3.2 Komentorivityökalut yleisesti 4
3.2.1 Top – resurssien monitorointi 5
3.2.2 Cat 5
3.2.3 Awk 6
3.2.4 Sed 6
3.2.5 Grep 7
3.2.6 Tail 8
4 Alustateknologiat 9
4.1 Kontit 9
4.2 Docker 9
4.3 Kubernetes 10
4.3.1 Cluster-arkkitehtuuri lyhyesti 10
4.3.2 Pod 11
4.3.3 Deployment 11
4.3.4 Ingress ja Gateway API 13
5 Monitorointityökalut 14
5.1 Grafana 14
5.1.1 Datalähteiden lisääminen 14
5.1.2 Näkymät 15
5.2 Prometheus 16
5.2.1 Toimintaperiaate 16
5.2.2 Tietomalli ja PromQL 17
6 Toteutus 19
6.1 Suunnittelu ja kehitys 19
6.1.1 Paikallinen kehitysympäristö 20
6.1.2 Tiedonkerääjä 23
6.1.3 Ohjelmistotestit 24

6.1.4 CI/CD-putki 26
6.2 Järjestelmän rakentaminen 29
6.2.1 Grafana 30
6.2.2 Prometheus 31
6.2.3 SSH-Poller 32
7 Yhteenveto ja pohdinta 34
Lähteet 35

1 Johdanto

Monitorointi on tärkeä osa valmistautumista vikatilanteiden torjumiseen ja niiden ennaltaehkäisyyn. Lisäksi se auttaa juurisyiden selvittämisessä hyödyntämällä ilmoituksista kerättyä historiallista dataa. Toistuvasti vikaantuvasta järjestelmästä jää yleensä merkkejä, joiden perusteella ongelman juurisyyt voidaan tunnistaa. Näitä merkkejä ovat esimerkiksi lokimerkinnät tai poikkeukselliset arvot järjestelmästä kerätyistä metriikoista vikaa edeltävinä hetkinä. Monitorointi mahdollistaa myös monien muiden tekijöiden epäsuoran seurannan metriikoiden avulla. Esimerkiksi palvelimen kuormaa voidaan arvioida suuntaa antavasti palvelimen sähkönkulutusta seuraamalla. Järjestelmien skaalautuessa niiden seuraaminen yksittäisen laitteen tasolla suoran yhteydenoton kautta käy mahdottomaksi.

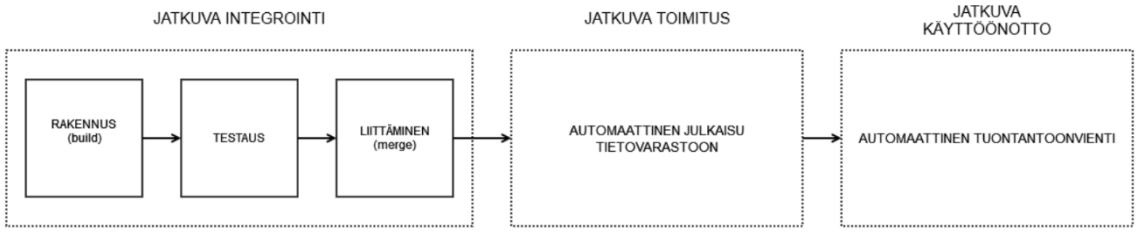
Opinnäytetyön toimeksiantajana toimii Elisa Oyj, jonka päätoimialana on langattomien verkkojen hallinta ja palvelut. Elisan liikevaihto vuonna 2024 oli 2.2 miljardia euroa. Sillä on maailmanlaajuisesti yli 6100 työntekijää ja sen pääkonttori sijaitsee Helsingissä (Elisa 2025). Opinnäytetyö toteutetaan Elisan Telco Services Automation and Tools -tiimissä, jonka vastuualue on telepalveluiden hallintajärjestelmien toiminnallisuus.

Tiimin järjestelmissä on lähtökohtaisesti mahdollista monitoroida metriikkatietoja sekä ilmoituksia, mutta tiedot ovat pirstaloituneina useisiin eri näkymiin, eikä keskitettyä näkymää järjestelmien yleistilanteesta ole. Työn tavoitteena on siis suunnitella ja toteuttaa pilotti (engl. Proof of Concept, PoC) keskitetystä valvontanäkymästä, josta tiimi voi tarkistaa palvelimien yleistilanteen. Järjestelmän ensimmäisessä versiossa palvelimista kerättävä data on rajattu järjestelmätason metriikkaan, joiden perusteella voidaan todentaa monitorointijärjestelmän käytännöllisyys. Jos järjestelmä todetaan toimivaksi, sitä voidaan jatkokehittää lisäämällä uusia ominaisuuksia ja datalähteitä myöhemmin. Opinnäytetyö keskittyy pääasiallisesti tiedon keräämisohjelmiston suunnitteluun yleistasolla, sekä käyttöönottoa varten tehtävän CI/CD-putken ja Kubernetes-ympäristön suunnitteluun ja rakentamiseen.

2 CI/CD: Jatkuva integrointi ja toimitus

2.1 CI/CD:n perusteet

CI/CD (engl. Continuos Integration, Continuous Delivery/Deployment) on sarja ohjelmistokehityskäytäntöjä, joilla pyritään sulavoittamaan ja nopeuttamaan ohjelmistojen kehityssykliä automatisoinnilla ja tihennetyllä koodin integrointivälillä. Samalla pyritään vähentämään tuotantoon asti pääsevien ohjelmistovikojen määrää. Hyödyt korostuvat, kun tiimit ja kehitettävät ohjelmistot kasvavat, kun manuaalinen koodin tarkastaminen ja toimitustyö muuttuu vaativammaksi. (Red Hat 2025.) Kuviossa 1 on ylätason kuvaus CI/CD-putken rakenteesta.



Kuvio 1. CI/CD-putken perusrakenne (mukailtu Red Hat 2025)

2.2 Jatkuva integrointi

Jatkuva integrointi (CI) on ensimmäinen osa CI/CD-putkea, joka koostuu automatisoidusta koodin varmistamisesta ennen Git-tietovaraston päähaaraan (engl. main), yhdistämistä. Nykyaikaisissa ohjelmistoprojekteissa on usein useita kehittäjiä kehittämässä eri ominaisuuksia samanaikaisesti eri kehityshaaroissa (engl. branch), joka lisää todennäköisyyttä keskenään ristiriitaisista muutoksista koodikantaan. Jatkuva integrointi vähentää todennäköisyyttä rikkoviin muutoksiin, koska kaiken päähaaraan liitettävän koodin täytyy läpäistä sarja automatisoituja yksikkö- ja integraatiotestejä. (Red Hat 2025.)

Ratkaisun toimivuuden määrittelee kuitenkin lopulta kirjoitettujen testien kattavuus, joka riippuu testien kehityksen huolellisuudesta. Usein CI/CD-putkesta vastaa kehittäjien kanssa yhteistyötä tekevä DevOps-tiimi, jolla varmistetaan, että ohjelmiston kehityksessä riittää resursseja myös sen varmistukseen ja toimituksen sujuvuuteen.

2.3 Jatkuva toimitus ja käyttöönotto

Jatkuva toimitus ja käyttöönotto (engl. Continuous Delivery, Continuous Deployment) ovat toinen osa CI/CD-putkea, jossa keskitytään automatisoimaan ja sujuvoittamaan ohjelmiston käyttöönottoa. Projektissa voidaan noudattaa jatkuvaa toimitusta tai käyttöönottoa - tai molempia. Molempien toiminta kuitenkin edellyttää, että jatkuva integraatio on toteutettu.

Jatkuvalla toimituksella tarkoitetaan automaattista koodipäivitysten toimittamista päähaaraan, jos puskettu päivitys läpäisee jatkuvan integraation automaattiset testit. Tämä varmistaa, että päähaara pysyy aina julkaisuvalmiina, ja toimii näin esiasteena jatkuvalle käyttöönotolle.

Jatkuva käyttöönotto on CI/CD-putken viimeinen osa, päähaara voidaan viedä tuotantoon. Tässä vaiheessa voidaan esimerkiksi kontittaa sovellus automaattisesti, ja viedä se konttirekisteriin, mistä tuotantoympäristö voi hakea päivitetyn ohjelmistokontin. Tämä mahdollistaa nopean käyttöönottovaiheen pienille muutoksille vähentäen julkaisuun liittyviä riskejä ja mahdollistaen paremman loppukäyttäjäpalautteen päivitysten yhteydessä.

3 Linux – komentorivityökalut

3.1 Proc-tiedostojärjestelmä

Proc-tiedostojärjestelmä on pseudo-tiedostojärjestelmä, mikä tarkoittaa, että sen sisältö ei ole kiintolevyllä, vaan sen tiedostoja päivitetään dynaamisesti järjestelmän muistissa. Se toimii käyttäjien ja ohjelmistojen rajapintana, josta voidaan hakea reaaliaikaista tietoa käyttöjärjestelmän tilasta, prosesseista ja laitteistosta. Esimerkiksi top-komento käyttää tätä järjestelmää tietolähteenään. Useimmat /proc-tiedostojärjestelmän resurssit ovat vain luettavissa, mutta sen kautta voidaan myös muokata joitakin kernel-muuttujia. (man7.org 2019b.)

3.2 Komentorivityökalut yleisesti

Lähes jokaisessa Linux-jakelussa on vakiotyökalut, joita voidaan hyödyntää järjestelmien automatisoinnissa ja monitoroinnissa. Niillä voidaan muun muassa hakea tietoa tietokoneen kuormasta tai käynnissä olevista prosesseista, muokata tekstitiedostoja tai käsitellä ohjelmien ja komentojen tulosteita. Komentorivityökalut ovat välttämätön osa palvelimien ylläpitoa, koska palvelinkoneissa ei yleensä ole graafista käyttöliittymää lainkaan. Usein komentorivityökalut noudattavat myös POSIX-standardia, mikä mahdollistaa laajan yhteensopivuuden eri Linux-jakeluiden välillä. Monet niistä toimivat myös muissa UNIX-pohjaisissa käyttöjärjestelmissä, kuten macOS:ssä (Thomas, T 2018).

Komentorivityökalut toimivat myös saumattomasti yhdessä mahdollistaen komentojen putkituksen (eng. piping). Putkittamisessa ensimmäisen komennon tuloste ohjataan toisen komennon syötteeksi. Näin voidaan tehdä monivaiheisia komentoketjuja yhdellä komentorivikäskyllä. Kuvassa 1 on esimerkki kolmivaiheisesta komentoputkesta.

Kuva 1. Esimerkki komentoputkesta

Esimerkin komentoputkessa ensimmäinen komento listaa aktiiviset prosessit, toinen suodattaa niistä prosessit, joissa esiintyy merkkijono ”ssh” ja kolmas laskee ja tulostaa rivien määrän, jota suodattamisen jälkeen on jäljellä. Seuraavissa alakappaleissa käsitellään yleisesti palvelimien automatisoinnissa ja monitoroinnissa hyödynnettäviä komentoja. Niiden toiminta selitetään yleistasolla ja esimerkinomaisesti.

3.2.1 Top – resurssien monitorointi

Top on niin kutsuttu TUI-sovellus (Text-based User Interface), joka palauttaa oletuksena terminaaliin interaktiivisen ja reaaliaikaisen kokonaiskuvan järjestelmän tilasta, sekä listan prosesseista ja kernelin käyttämistä säikeistä. Sen toimintaa voidaan kuitenkin muokata konfiguraatiotiedostoilla sekä komentoriviasetuksilla, esimerkiksi rajaamaan annettuja tietoja tai vaihtamaan tulostettuja yksiköitä. Topin kaltaisia sovelluksia on useita, kuten esimerkiksi htop tai btop, mutta top löytyy esiasennettuna muita vaihtoehtoja todennäköisemmin (Sharma, S 2022). Kuvassa 2 on topin käyttöliittymä.

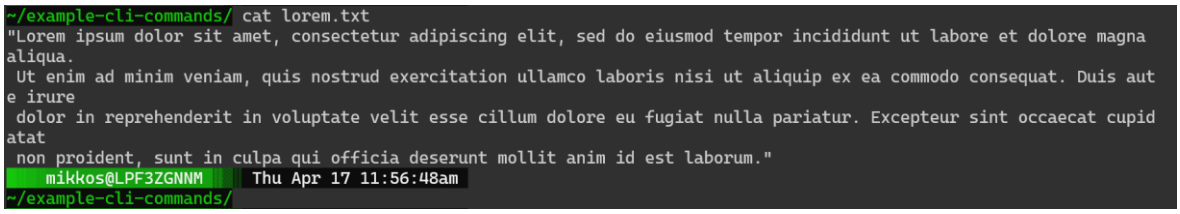
```
top - 11:53:11 up 0 min,  1 user,  load average: 0.28, 0.09, 0.03
Tasks:  20 total,   1 running,  19 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem :   7778.3 total,   7028.8 free,    646.0 used,    332.0 buff/cache
MiB Swap:   2048.0 total,   2048.0 free,      0.0 used.   7132.3 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
      1 root      20   0  167500  11896   9012 S   0.0   0.1   0:00.35 systemd
      2 root      20   0    2776   1924   1796 S   0.0   0.0   0:00.00 init-systemd(De
      6 root      20   0    2776      4      0 S   0.0   0.0   0:00.00 init
     35 root      20   0   41132  15304  14224 S   0.0   0.2   0:00.08 systemd-journal
     50 root      20   0   23760   5736   4500 S   0.0   0.1   0:00.11 systemd-udevd
    135 root      20   0    6612   1188   1048 S   0.0   0.0   0:00.00 cron
    136 message+  20   0    7912   3616   3248 S   0.0   0.0   0:00.01 dbus-daemon
    138 root      20   0   16716   7836   6824 S   0.0   0.1   0:00.05 systemd-logind
    144 root      20   0 1870372  44808  28352 S   0.0   0.6   0:00.15 containerd
    145 root      20   0    5500   1036    948 S   0.0   0.0   0:00.00 agetty
    146 root      20   0    5876   1004    916 S   0.0   0.0   0:00.00 agetty
    168 root      20   0 2273372  83652  46188 S   0.0   1.1   0:00.51 dockerd
    762 root      20   0    2784    208     80 S   0.0   0.0   0:00.00 SessionLeader
    763 root      20   0    2784    212     80 S   0.0   0.0   0:00.00 Relay(764)
    764 mikkos    20   0    9888   5836   4096 S   0.0   0.1   0:00.04 zsh
    765 root      20   0    5800   3544   3072 S   0.0   0.0   0:00.00 login
    771 mikkos    20   0   19088  10724   8892 S   0.0   0.1   0:00.04 systemd
    772 mikkos    20   0  168240   2992      0 S   0.0   0.0   0:00.00 (sd-pam)
    787 mikkos    20   0    9368   5292   4084 S   0.0   0.1   0:00.02 zsh
    794 mikkos    20   0   11624   4932   3048 R   0.0   0.1   0:00.00 top
```

Kuva 2. Top-käyttöliittymä

Esimerkissä on top-sovelluksen käyttöliittymä vakioasetuksilla. Käyttöliittymän yläosassa on käyttöjärjestelmätason yhteenveto kirjautuneista käyttäjistä, prosesseista, sekä prosessorin ja muistin käytöstä. Sen alla on lista, josta käyttäjä voi tarkistaa mitkä prosessit kuluttavat resursseja.

3.2.2 Cat

Cat-komennon nimi tulee sanasta concatenate, eli ketjuttaa. Komento siis ketjuttaa yhden tai useamman tiedoston sisällön ja tulostaa sen (man7.org 2025b). Kuvassa 3 on esimerkki cat-komennon käytöstä tekstitiedoston lukemiseen.

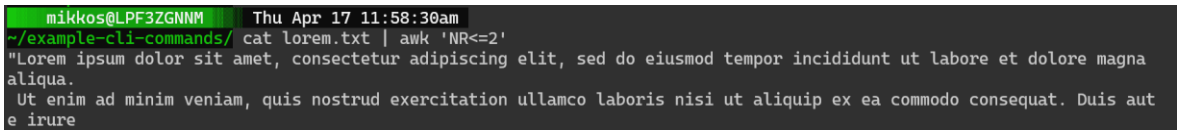


Kuva 3. Esimerkki cat-komennon käytöstä

Yleinen käyttötarkoitus cat-komennolle on kuitenkin tulostaa yhden tiedoston sisältö, joko sellaisenaan tai osana komentoputkea, jolloin sisältö ohjataan toisen komennon käsiteltäväksi tai esimerkiksi toiseen tiedostoon (man7.org 2025b).

3.2.3 Awk

Awk on tekstin kuvioiden tunnistukseen sekä prosessointiin tarkoitettu ohjelmointikieli. Se toimii myös komentorivityökaluna Unix- ja Linux-järjestelmissä, ja sitä voidaan käyttää ohjelmien tulosteiden sekä tekstipohjaisten tiedostojen rajaamiseen ja muokkaamiseen. (man7.org 2025a.) Kuvassa 4 on esimerkki awk-komennon käytöstä tulosteen rajaamiseen.

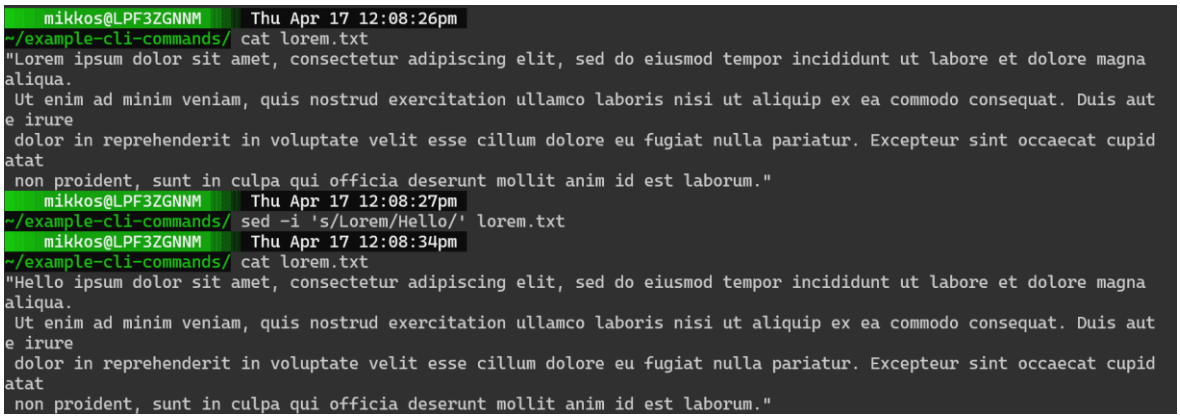


Kuva 4. Esimerkki awk-komennon käytöstä

Kuvan 4 esimerkissä awk-komentoa käytetään putkitettuna cat-komentoon lorem.txt-tiedoston kahden ensimmäisen rivin tulostamiseen. Awk-komennolle annetaan parametrina ”NR<=2”, jossa ”NR” tarkoittaa rivin numeroa. Määritelty ehto on siis tulostaa tekstirivi, jos sen numero on vähemmän tai yhtä suuri kuin 2. Saman lopputuloksen saisi käyttämällä komentoa head –2, mutta awk-komennon hyöty tulee esille, kun haettavan tai muokattavan tekstin tarvittava ehto vaatii tarkempaa määrittelyä.

3.2.4 Sed

Sed-komennon nimi tulee sanoista stream editor. Sitä käytetään tekstitiedostojen tai ohjelmien tekstimuotoisen tulosteen jäsentelyyn, muokkaamiseen ja rajaamiseen (man7.org 2017). Sen yleisimpiä käyttötarkoituksia on etsiä ja korvata tekstistä sanoja, tai lisätä tekstiä, esimerkiksi konfiguraatiotiedostoihin. Sed on usein myös osana komentoputkea. Kuvassa 5 on esimerkki sed-komennon käytöstä sanan etsimiseen ja korvaamiseen.

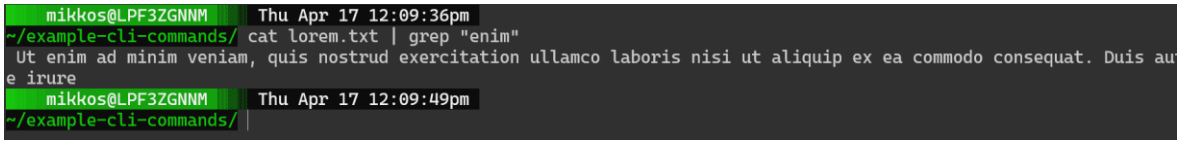


Kuva 5. Esimerkki sed-komennon käytöstä

Yllä olevassa esimerkissä Lorem.txt tiedostossa etsitään sana ”Lorem” ja se korvataan sanalla ”Hello”. Käytännössä tätä voisi käyttää esimerkiksi toistuvan muuttujan nimen vaihtamiseen kooditiedostossa.

3.2.5 Grep

Grep-työkalulla voidaan hakea tulostetekstistä rivejä, jotka vastaavat komennolle parametrina annettua hakukuviota. Hakukuvio voi olla merkkijonokuvio, jota tekstistä etsitään, tai säännönmukainen lauseke (engl. regular expression). (man7.org 2019a.)



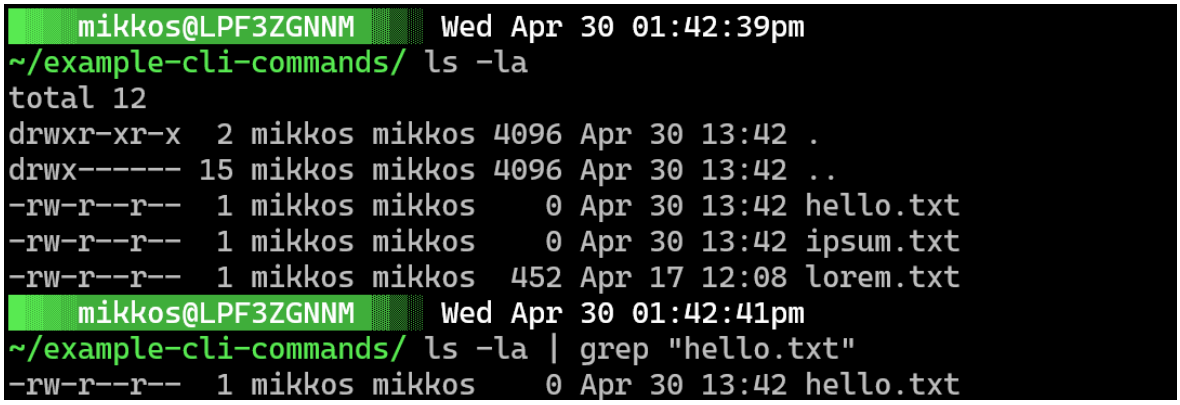
Kuva 6. Grep-komento osana putkea

Grep on usein osa putkea, jolla etsitään komennon tulosteesta haluttu osa tekstiä. Yllä olevassa esimerkissä etsitään cat-komennon tulosteesta riviä, josta löytyy merkkijono ”enim”. Sitä voidaan kuitenkin käyttää myös itsenäisesti, jos tekstilähteenä toimii tiedosto, kuten kuvan 7 esimerkissä.

Kuva 7. Grep-komennon käyttö itsenäisesti tiedostosta hakukuvion etsimiseen

Yllä oleva grep-komento palauttaa saman tuloksen kuin aikaisempi cat-komennon kanssa putkitettu komento, koska molemmat tulostavat lorem.txt tiedoston sisältöä. Komennot eroavat toisistaan lievästi siten, että cat-komennon kanssa putkitettu grep-komento etsii

tulosteesta halutut rivit, kun taas itsenäisesti kutsuttuna se etsii rivit suoraan halutusta tiedostosta. Kuvassa 8 on käytännönläheinen esimerkki grep-komennon putkittamisesta.

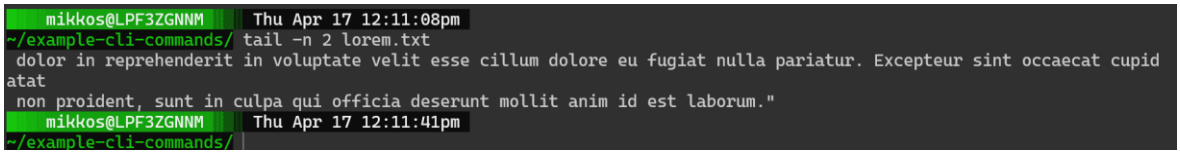


Kuva 8. Esimerkki grep-komennon käytöstä muuhun kuin tekstitiedostojen lukuun

Komento ls -la listaa komentoriville kaikki aktiivisen hakemiston sisältämät tiedostot. Esimerkissä niitä on kolme, mutta hakemistojen kasvaessa tietyn tiedoston löytäminen muuttuu haastavammaksi. Jos ls-komennon kuitenkin putkittaa grep-komentoon, voidaan nopeasti tarkistaa, onko haluttu tiedosto hakemistossa.

3.2.6 Tail

Tail-komennolla voidaan tulostaa haluttu määrä rivejä tiedoston tai komennon tulosteen lopusta. (man7.org 2025c). Sille on myös vastaava komento head, jolla voidaan vastaavasti hakea haluttu määrä rivejä tekstin alkuosasta. Kuvassa 9 on esimerkki tail-komennon käytöstä.



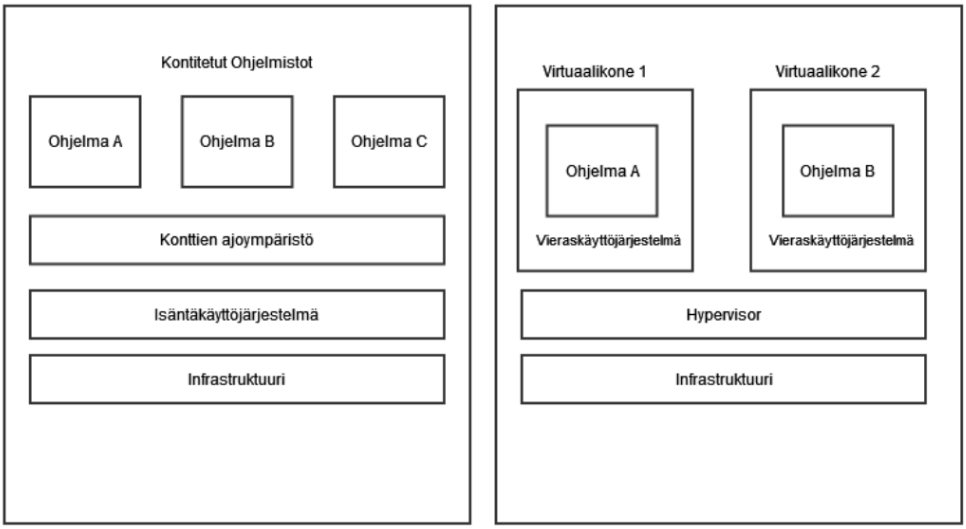
Kuva 9. Esimerkki tail komennosta kahden viimeisen rivin hakemiseen

Kuten grep- tai awk-komennoissa, tail-komentoa voidaan käyttää joko itsenäisesti tai putkitettuna käyttötilanteen mukaan. Sitä voidaan käyttää esimerkiksi viimeisimpien tapahtumien hakemiseen ohjelman lokitulosteista.

4 Alustateknologiat

4.1 Kontit

Kontit (eng. Container) ovat tapa ajaa ohjelmistoja omissa eristetyissä ympäristöissä, joihin asennetaan kaikki ohjelmiston vaatimat riippuvuudet. Kontit rakennetaan konttikuvan (eng. container image) pohjalta, jossa riippuvuudet ja pohjakäyttöjärjestelmä määritellään. Kuvien avulla identtisten ajoympäristöjen luominen kehitystä ja käyttöönottoa varten helpottuu. Kuviossa 2 verrataan ohjelmistojen ajamista konteissa perinteiseen virtualisointiin.



Kuvio 2. Kontitetut ohjelmistot verrattuna perinteisiin virtuaalikoneisiin (mukailtu Docker 2025b)

Toisin kuin perinteisessä virtualisoinnissa, konteissa virtualisoidaan käyttöjärjestelmän toimintaa laitteiston sijaan. Jos isäntäkäyttöjärjestelmä on Linux, kontit jakavat isäntäkäyttöjärjestelmän kernelin, mikä tekee ratkaisusta kevyemmän ja tehokkaamman. Hypervisorin sijaan konteilla on oma ajoympäristönsä, kuten Dockerin tapauksessa kontit ajetaan Docker Enginen päällä. (Docker 2025b.)

Windows- ja macOS-järjestelmissä kernelin jakaminen ei ole mahdollista natiivisti, joten konttialusta virtualisoi kevyen version Linux-kernelistä konttien luomisen yhteydessä. Tässäkin tapauksessa ohjelmistojen ajaminen konteissa on merkittävästi vähemmän resursseja vaativaa verrattuna perinteisiin virtuaalikoneisiin. (Docker 2025b; Buchanan, I 2025.)

4.2 Docker

Konttiteknologioita on useita, ja Docker on yksi käytetyimmistä (StackOverflow 2024). Se tarjoaa sekä graafisen että komentorivipohjaisen käyttöliittymän konttien luomiseen ja

hallintaan, konttien ajoympäristön (Docker engine), sekä kuvarekisterin (Docker hub), josta voidaan hakea valmiita kuvia useisiin eri käyttötarkoituksiin.

Dockerilla on omat konfigurointikäytäntönsä konttikuvien luomiselle, Dockerfilet, joiden avulla voidaan helposti määritellä tarvittavat riippuvuudet ja pohjakuva ajettavalle kontille. Lisäksi Docker tarjoaa ratkaisun konttiryhmien luomiselle YAML-tiedostoja käyttämällä, Docker compose.

4.3 Kubernetes

Kubernetes on konttien orkestrointijärjestelmä, jolla voidaan hallita, skaalata ja käyttöönottaa kontitettuja ohjelmistoja. Google kehitti sen omien konttihallintatyökalujensa pohjalta, ja heidän päätavoitteensa oli helpottaa monimutkaisten hajautettujen konttipohjaisten järjestelmien hallintaa. Google julkaisi järjestelmän avoimen lähdekoodin ohjelmistona vuonna 2014, ja sen kehitys- ja isännöintivastuu siirtyi nopeasti sen jälkeen Cloud Native Computing Foundation-järjestön vastuulle. (Burns, B. & Grant, B. & Oppenheimer, D. & Brewer, E. & Wilkes, J. 2016.)

Tuotantoympäristössä palveluiden tulee olla aina saatavilla, joten kaikkia kontteja tulee ylläpitää jatkuvasti. Kuberneteksen tehtävä on helpottaa niiden ylläpitoa automatisoidulla hallinnalla, sekä tarjoamalla kehyksen konttiympäristön konfiguroinnin selkeyttämiseksi. Kubernetes tarjoaa automatisoinnin esimerkiksi poistettujen konttien korvaamiseen uusilla konteilla, kontin tilan hallintaan sekä itsekorjautuvuuteen. Resurssien määrittely tapahtuu YAML-tiedostoilla, ja niiden käyttöönotto tapahtuu Kubernetes API:n kautta. Rajapinnan kanssa kommunikointiin on vaihtoehtoisia komentorivityökaluja, kuten esimerkiksi kubectl tai kubeadm. (Kubernetes 2025e.)

Kubernetes koostuu useista erilaisista resurssityypeistä, joilla palveluita voidaan hallita, skaalata ja verkottaa. Seuraavissa alakappaleissa käsitellään keskeisiä resurssityyppejä, jotka toimivat pohjana valtaosalle Kubernetes-pohjaisista ratkaisuista.

4.3.1 Cluster-arkkitehtuuri lyhyesti

Yksittäistä Kubernetesympäristöä kutsutaan klusteriksi, ja se koostuu kontrollitasosta (engl. Control plane) sekä nodeista, joihin Kuberneteksessä ajettavat kontit sijoitetaan. Kontrollitaso hallinnoi koko klusterin toimintaa, sekä vastaa ylätason tehtävien kuten ajoituksen tai tilan (engl. state) hallinnasta. Nodet ovat fyysisiä tai virtuaalisia palvelimia, joilla varsinaisesta työkuormasta vastaavat kontit ajetaan. (Kubernetes 2025a.)

4.3.2 Pod

Pod on Kubernetes-ekosysteemin pienin resurssiyksikkö. Sillä tarkoitetaan yhtä tai joukkoa kontteja, joilla on jaetut säilytys- ja tietoverkkoresurssit, sekä yhteinen määrittely ajamista varten. Kuvassa 10 on esimerkki podien määrittelyyn tarkoitetusta YAML-tiedostosta.

```yaml
apiVersion: v1
kind: Pod
metadata:
  namespace: demo
  name: demo-pod
  labels:
    app: demo
spec:
  containers:
  - image: busybox
    name: BusyBox
    command: ["sleep", "3600"]
```

Kuva 10. Busybox-podin konfigurointitiedosto

Yleinen tapa käyttää podeja on antaa yksi kontti yhden podin hallittavaksi, mutta tässäkään tapauksessa Kubernetes ei hallitse konttia suoraan, vaan podia, joka toimii sen kuorena (eng. wrapper). (Kubernetes 2025f.)

4.3.3 Deployment

Deployment-resurssit ovat tarkoitettu Pod- ja ReplicaSet-resurssien pitämiseen halutussa tilassa, sekä niiden yleiseen ylläpitoon. Podeja ei yleensä ajeta itsenäisesti, vaan ne ovat useimmiten Deployment-resurssien hallinnassa, koska Deployment vastaa podien automaattisesta luomisesta, päivittämisestä ja skaalaamisesta sille tehdyn määrittelyn mukaisesti. (Kubernetes 2025b.) Kuvassa 11 on esimerkki BusyBox-deploymentin määrittelystä.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  namespace: demo
  name: demo-busybox
  labels:
    app: demo-busybox
spec:
  replicas: 1
  selector:
    matchLabels:
      app: demo-busybox
  template:
    metadata:
      labels:
        app: demo-busybox
    spec:
      containers:
        - image: busybox
          name: demo-busybox
          command: ["sleep", "3600"]
```

Kuva 11. Busybox-deploymentin määrittelytiedosto

Service-resurssit mahdollistavat Deploymenteissa ajettavien ohjelmien asettamisen näkyville Kubernetes-Klusterin sisäverkossa. Oletusarvoisesti Service-resurssi ei kuitenkaan julkaise palvelua julkisesti internetiin tai yrityksen sisäverkkoon. Ulkoiseen liikenteeseen ja sen reititykseen tarvitaan erillinen resurssi, kuten Ingress, Load Balancer tai Gateway-resurssi. (Kubernetes 2025f.) Kuvassa 12 on esimerkki yksinkertaisen Service-resurssin määrittelytiedostosta.

```yaml
apiVersion: v1
kind: Service
metadata:
  namespace: demo
  name: demo-busybox-service
  labels:
    app: demo-busybox
spec:
  selector:
    app: demo-busybox
  ports:
  - protocol: TCP
    port: 80
    targetPort: 8080
```

Kuva 12. Esimerkki service-resurssin määrittelytiedostosta

4.3.4 Ingress ja Gateway API

Ingress on resurssi, jolla HTTP- ja HTTPS-pohjainen liikenne voidaan reitittää, jossa reitityssäännöt määritellään resurssin konfiguraatiotiedostossa. Ingress-resurssityyppiä käytetään edelleen ja useimmat Kubenetes-klusterit tukevat sitä, mutta sen aktiivinen kehitys on lopetettu. Sen tilalle on kehitetty Gateway API, joka on suunniteltu korvaamaan Ingress-resurssin käytön. (Kubernetes 2025c.)

Gateway API on uudempi reititysstandardi, joka tukee myös muita protokollia HTTP:n ja HTTPS:n lisäksi. Gateway API koostuu kolmesta pääasiallisesta Kind-resurssista: Gateway, HTTPRoute ja GatewayClass. Kind-resurssien avulla voidaan hallita verkkoliikenteen reititystä Kubernetes-Klusterin ulko- ja sisäpuolella. Kolmen pohjaresurssin lisäksi Gateway API sisältää myös muita Kind-resursseja, joita käytetään esimerkiksi muiden protokollien reititykseen.

Gateway-resurssi määrittelee verkkorajapinnan, eli mitä osoitetta ja porttia kuunnellaan. HTTPRoute puolestaan määrittelee palvelun API-reitityksen, eli kuunneltavat päätepisteet kuten esimerkiksi /home tai /users. GatewayClass-resurssilla voidaan eksplisiittisesti määritellä verkkorajapinnan toteutustapa, kuten esimerkiksi mahdollinen kuormantasaaja tai web-palvelin.

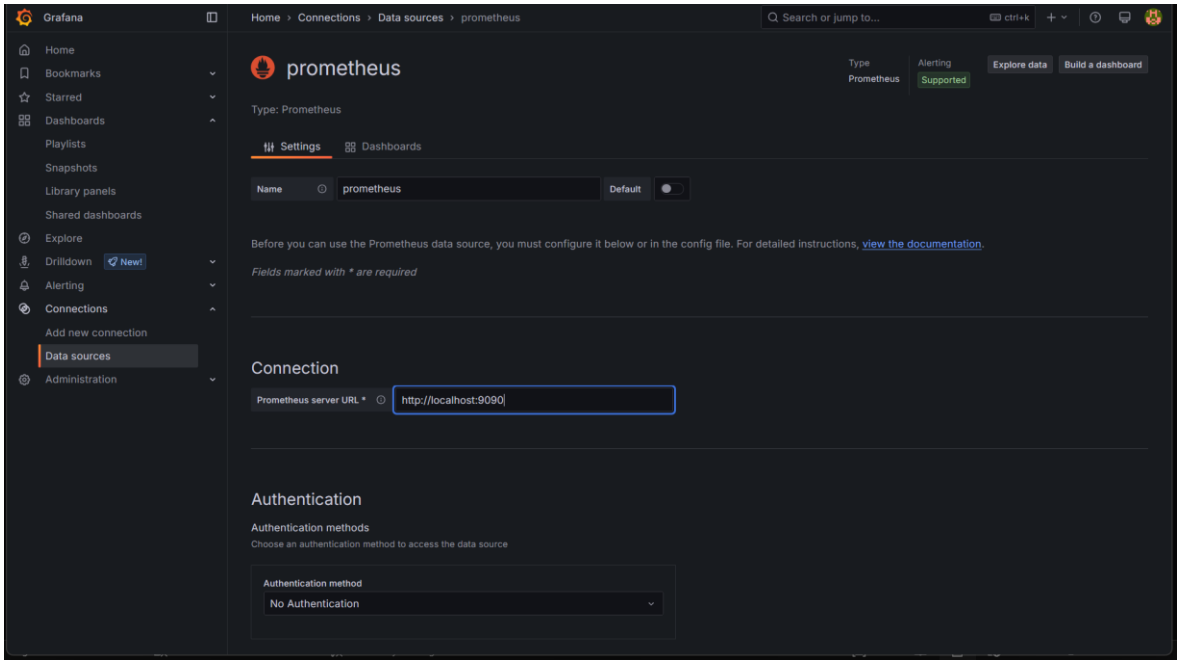
5 Monitorointityökalut

5.1 Grafana

Grafana on visualisointi- ja analytiikka-alusta aikasarjamuotoiselle datalle (Grafana 2025). Sillä voidaan toteuttaa näkymiä, jossa monitorointidataa voidaan seurata tietyllä aikavälillä. Lisäksi datan arvoille voidaan konfiguroida ilmoituksia, jos ne ylittävät tai alittavat tietyn arvon. Grafana tallentaa dataa määritellylle aikavälille seurantaa varten, mutta integroitua tietokantaa ei ole tarkoitettu pysyvään datan säilytykseen. Pysyvään tallennukseen on kuitenkin vaihtoehtona esimerkiksi Grafanan oma ratkaisu Loki, tai muita kolmannen osapuolen ratkaisuja.

5.1.1 Datalähteiden lisääminen

Grafana tukee useita erityyppisiä datalähteitä. Kyseessä voi olla esimerkiksi SQL- tai Loki-tietokanta, Prometheus-metriikankerääjä tai HTTP-pohjainen API. Tietolähteen lisäämiseen on oma graafinen valikkonsa (kuva 13).

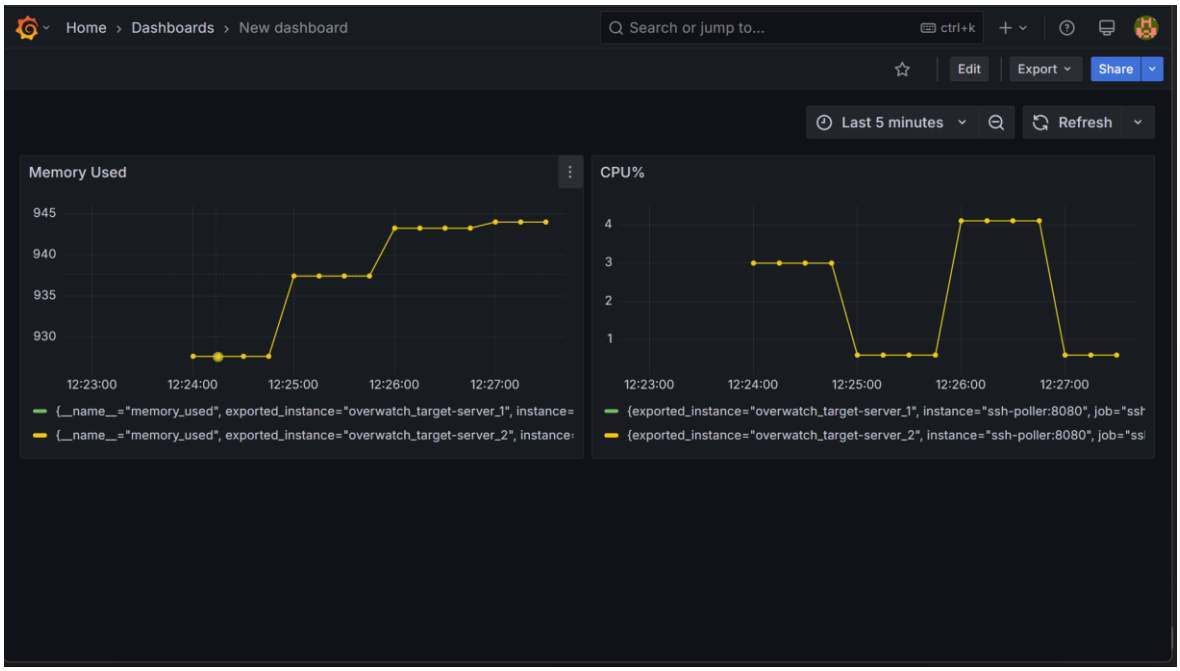


Kuva 13: Datalähteen lisäämisvalikko Grafanassa

Datalähteen saa lisättyä valitsemalla lähteen tyypin, kuten yllä olevassa kuvakaappauksessa on valittuna Prometheus, jonka jälkeen tehdään lähdekohtainen konfigurointi, jossa lisätään muun muassa tietolähteen osoite.

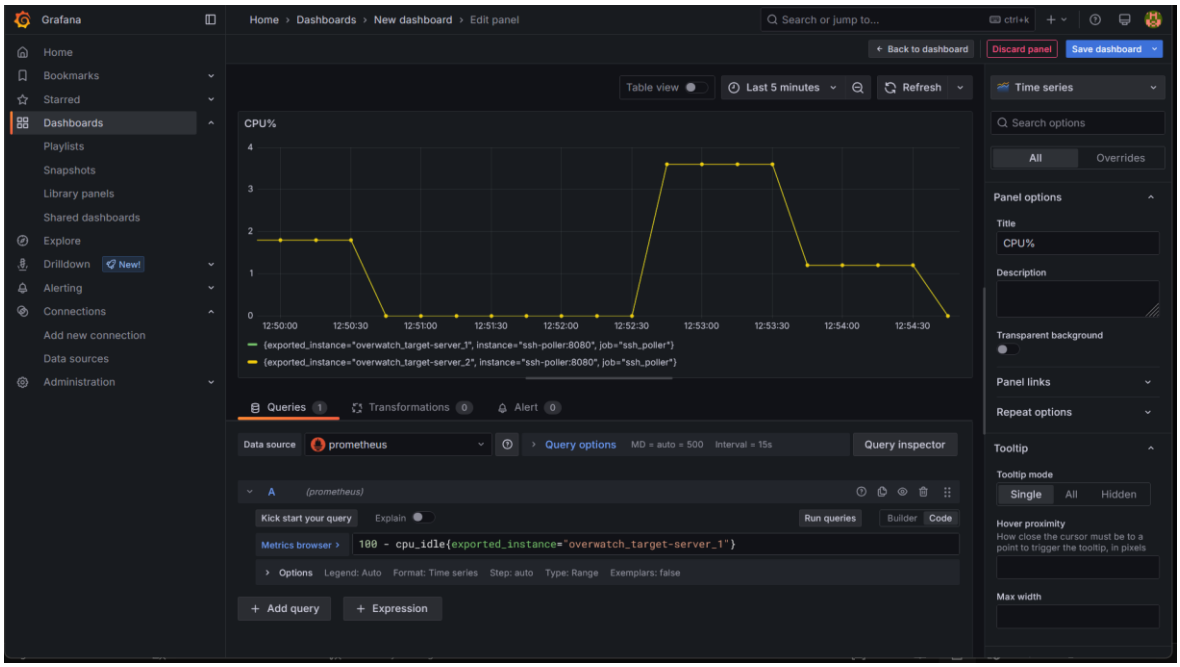
5.1.2 Näkymät

Grafanalla voidaan koota kasaan järjestelmän tai muuten toisiinsa liittyviä metriikoita näkymiin, josta saa yleiskuvan vallitsevasta tilanteesta. Tietoja voidaan esittää erilaisina kuvaajina tai yksittäisinä arvoina, ja näkymiin voidaan myös lisätä esimerkiksi Markdown- tai HTML-muotoista dataa. Kuvassa 14 on esimerkki yksinkertaisesta näkymästä, jossa on kaksi kuvaajaa.



Kuva 14. Esimerkki näkymästä

Grafana tarjoaa myös tietolähdekohtaisen kyselyeditorin, jota voidaan hyödyntää, kun näkymään lisätään kuvaajia. Kuvassa 15 on esimerkki, jossa kyselyeditorilla haetaan Prometheus-palvelimelle kerättyä prosessorin kuormametriikka PromQL-kielellä (Prometheus Query Language).



Kuva 15. Kuvaajan lisääminen Grafanaan

PromQL mahdollistaa myös kyselynsisäiset laskutoimitukset. Yllä olevassa esimerkissä toteutetaan kuvaaja prosessorin kuormasta vähentämällä prosessorin prosentuaalinen taukoaika järjestelmän kokonaisajasta.

5.2 Prometheus

Prometheus on monitorointi- ja ilmoitustyökalu, jolla voidaan kerätä aikasarjamuotoista metriikkatietoa. Sen kehitti SoundCloud vuonna 2012, mutta se avattiin avoimen lähdekoodin ohjelmistoksi ja sen hallinnointi jätettiin Cloud Native Computing Foundationin vastuulle vuonna 2016. (Prometheus 2025a.)

5.2.1 Toimintaperiaate

Prometheus toimii käyttäen pull-mallia, jossa data kerätään (engl. scraping) valvottavien kohteiden tarjoamista /metrics-päätepisteistä. Dataa kerätään Prometheuksen konfiguraatiotiedostossa määritellyin väliajoin, ja se tallennetaan Prometheuksen omaan väliaikaiseen säilöön. Säilöön kerätty data on aikasarjamuotoista dataa, jota voidaan hakea kolmannen osapuolen sovelluksilla tai PromQL-kielellä Prometheuksesta jatkokäyttöä varten.

Kuva 16. Esimerkki Prometheus-konfiguraatiotiedostosta

Kuvan 16 esimerkkikonfiguraatiossa Prometheus-palvelin on asetettu keräämään dataa 60 sekunnin välein Docker-kontista, jonka osoite Dockerin sisäisessä verkossa on ”target:8080”. Prometheuksen on myös tarkoitus toimia lähes autonomisesti, kun sen asetukset on ensin määritelty konfiguraatiotiedossa.

5.2.2 Tietomalli ja PromQL

Prometheus käyttää omaa aikasarjadata-malliaan, jossa metriikka koostuu sen nimestä, ja vapaaehtoisista avain-arvo-pareista, joita kutsutaan leimoiksi (engl. labels), sekä metriikan arvosta, joka on esitetty float64-muodossa. (Prometheus 2025.) Kuvan 17 esimerkissä on Go:n promhttp-kirjaston vakiona palauttamaa dataa go-ohjelman roskienkerääjän aktiivisuusajasta.

```
# HELP go_gc_duration_seconds A summary of the wall-time pause (stop-the-world) duration in garbage collection cycles.
# TYPE go_gc_duration_seconds summary
go_gc_duration_seconds{quantile="0"} 0
go_gc_duration_seconds{quantile="0.25"} 0
go_gc_duration_seconds{quantile="0.5"} 0
go_gc_duration_seconds{quantile="0.75"} 0
go_gc_duration_seconds{quantile="1"} 0
go_gc_duration_seconds_sum 0
go_gc_duration_seconds_count 0
```

Kuva 17. Esimerkki Prometheus-standardin mukaisen päätepisteen palauttamasta metriikkadatasta

Metriikan ensimmäinen arvo, kuten go_gc_duration_seconds on metriikan nimi. Toinen arvo on leima kuten quantile, jolla haettavaa tietoa voidaan rajata. Esimerkissä roskienkerääjän aktiivisuustieto voidaan rajata esimerkiksi mediaaniin (0,5) tai maksimiin (1). Viimeinen arvo on itse metriikan arvo, joka esimerkissä on 0.

Prometheus tarjoaa neljä metriikkatyyppiä: laskuri (counter), mittari (gauge), histogrammi (histogram) ja tiivistelmä (summary). Prometheus-instanssit eivät kuitenkaan itse hyödynnä näitä tyyppejä, vaan ne on tarkoitettu pääasiassa asiakaskirjastoille (engl. client library),

jotta ne voivat tarjota sopivan ohjelmistorajapinnan kunkin metriikkatyypin käyttötarkoitukseen. Prometheus-palvelin käsittelee metriikkadatan kuitenkin tyypittämättömässä muodossa. (Prometheus 2025b.)

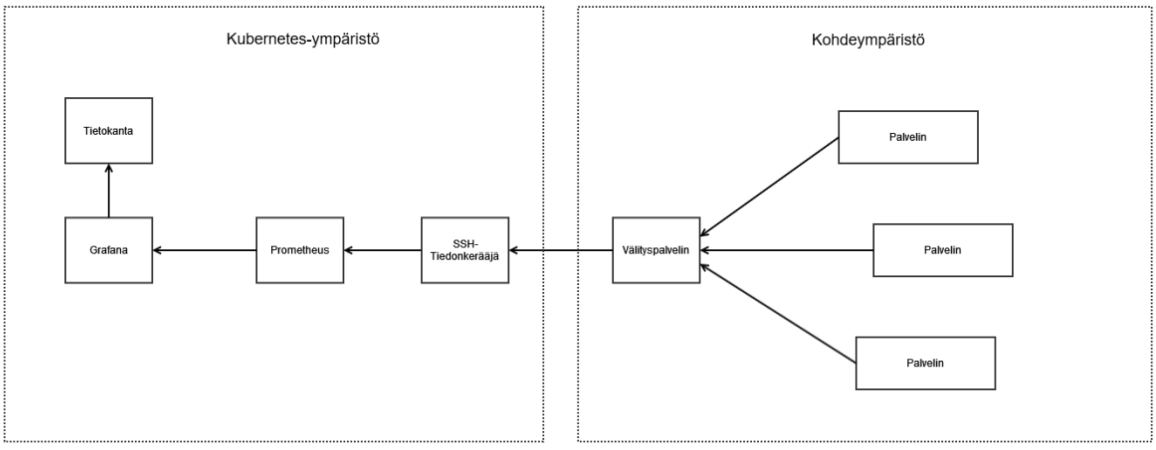
6 Toteutus

6.1 Suunnittelu ja kehitys

Monitorointijärjestelmällä on selkeät vaatimukset. Datan visualisointiin käytetään Grafanaa, jossa tietolähteenä toimii Prometheus. Prometheus tarvitsee itselleen sopivan tietolähteen, eikä se pysty keräämään tietoja suoraan palvelimilta. Vaihtoehdot tietolähteelle on joko kolmannen osapuolen tietojenkerääjä (engl. metrics exporter) tai itse toteutettu logiikka, joka vastaa Prometheuksen API-standardia.

Monitoroitavien palvelimien palveluntarjoaja ei kuitenkaan salli kolmannen osapuolen ohjelmistojen asentamista palvelimille, joten tiedonkeruun täytyy tapahtua palvelimien ulkopuolelta. Tähän tarkoitukseen ei löytynyt luotettavaa valmista ohjelmistoa, joten logiikka täytyi toteuttaa itse.

Suunnitelmana oli siis rakentaa SSH-pohjainen väliohjelmisto, joka ottaa yhteyden palvelimiin määritellyin väliajoin, kerää tiedot, ja palauttaa ne Prometheukselle sopivassa formaatissa. Ohjelmointikieleksi valittiin Go, koska järjestelmä vaatii kommunikointia useiden palvelimien kanssa, jolloin Go:n tarjoama tuki rinnakkaisuudelle sekä sen yleinen suorituskyky ovat hyödyllisiä. Lisäksi kyseessä on korkean tason kieli minimalistisella syntaksilla, joka on helppo oppia vaikka pääasiallinen työnkuva ei olisikaan ohjelmointipainotteista. Tämä on tärkeää, koska tiimissä työskentelee pääasiassa järjestelmänvalvojia. Kuviossa 3 on ylätason kuvaus suunnitellusta ympäristöstä ja sen kohdeympäristöstä.



Kuvio 3. Suunniteltu järjestelmän rakenne

Kohdeympäristössä SSH-liikenne kulkee yhden SSH-välityspalvelimen (engl. jump-box) kautta muille palvelimille. Suunnitelma on ylläpitää pysyvä SSH-yhteys välityspalvelimen ja väliohjelmiston välillä, ja muodostaa yhteys muihin palvelimiin 60 sekunnin välein

tiedonkeruuta varten. Järjestelmän myöhemmässä käyttöönottovaiheessa tullaan seuraamaan erityisesti välityspalvelimen kuormaa ja tekemään muutoksia tarvittaessa. Tarvittaessa yhteyksien muodostusta voidaan porrastaa, jotta välityspalvelimen ei tarvitse muodostaa samanaikaista yhteyttä jokaiseen palvelimeen. Tiedonkeruuväliä voidaan myös rajatuissa määrin pidentää, mutta se vaikuttaa kykyyn vastata vikatilanteisiin. Tämän takia maksimitiedonkeruuväliksi määriteltiin 5 minuuttia.

Lopullinen järjestelmä tulee toimimaan KaaS-ympäristössä (Kubernetes-as-a-Service). Kehittämällä ohjelmistoa suoraan kohdeympäristöön olisi mahdollista välttää käyttöönottoon liittyviä ongelmia, mutta olisi kokonaisuudessaan työlästä. Tämän takia kehitystä varten täytyi rakentaa konttipohjainen testiympäristö, jossa ohjelmistoa voidaan testata helpommin. Tämän lisäksi käyttöönoton helpottamiseksi rakennettiin välivaihe Github Actions – toimintojen avulla, jossa ohjelmisto testataan, kontitetaan (engl. Containerize) ja siirretään konttirekisteriin Kubernetes-alustalle käyttöönottoa varten.

6.1.1 Paikallinen kehitysympäristö

Paikallista kehitysympäristöä rakennettaessa hyödynnettiin valmiita kontteja osiin, joita ei itse kirjoiteta. Grafanalle ja Prometheukselle löytyy valmiit Docker-kuvat Docker Hubista. SSH-tiedonkerääjää ja kohdepalvelimia varten täytyi rakentaa omat kuvat. Pohjakuvana olisi voitu käyttää resurssien säästämiseksi kevyttä Linux-jakelua, kuten Alpine Linuxia, mutta SSH-konfiguroinnin helpottamiseksi pohjakuvaksi valittiin Ubuntu (kuva 18).

```
8   # Minimal container for running the program
7   FROM ubuntu:latest
6
5   RUN apt-get update && apt-get install -y openssh-client
4
3   COPY .env /app/.env
2   COPY ./bin/ssh-poller /app/ssh-poller
1   WORKDIR /app

1   EXPOSE $METRICS_PORT
2
3   CMD ["sh", "-c", "/app/ssh-poller"]
4
5
```

Kuva 18. Tiedonkerääjän ajoympäristön määrittelytiedosto

Tiedonkerääjän ajoympäristö ei tarvitse muita lisäpaketteja kuin OpenSSH-asiakasohjelmiston, jotta se voi ottaa yhteyden kohdepalvelimiin. Go-koodi kääntyy binäärimuotoon, joten se voidaan kääntää isäntäkoneella ja kopioida kuvan rakennusvaiheessa kuvaan. Näin konttiin ei tarvitse asentaa Go:ta eikä sen ajoympäristöä (engl. runtime). Tämän lisäksi isäntäkoneelta kuvaan kopioidaan ympäristömuuttujat. Ympäristömuuttujien kokoaminen .env-tiedostoon on väliaikainen ratkaisu kehitysvaiheeseen. KaaS-ympäristöä varten valitaan tietoturvallisempi ratkaisu. Lisäksi kuvalle avataan ympäristömuuttujissa määritelty portti, josta Prometheus voi kerätä ohjelmiston järjestelmätiedot. Seuraavaksi määriteltiin kohdepalvelinkontit (kuva 19).

```
FROM ubuntu:latest

# Test container for local development with SSH-server installed
ARG PW USER

ENV DEBIAN_FRONTEND=noninteractive

# Update + install & configure SSH Server
RUN apt-get update && apt-get install -y openssh-server  && \
    mkdir /var/run/sshd && \
    echo "${USER}:${PW}" | chpasswd && \
    sed -i "s/#PermitRootLogin prohibit-password/PermitRootLogin yes/" /etc/ssh/sshd_config && \
    echo "PasswordAuthentication yes" >> /etc/ssh/sshd_config

# Expose SSH port
EXPOSE 22

# Start ssh service
CMD ["/usr/sbin/sshd", "-D"]
```

Kuva 19. Kohdepalvelimen kuvan määrittely

Kohdepalvelimelle täytyi asentaa ja konfiguroida OpenSSH-palvelin. Konfiguroinnissa muutettiin SSH-käyttäjän salasana, sallittiin juuritason SSH-kirjautuminen sekä sallittiin kirjautuminen käyttäen salasanaa. Konfigurointitiedostojen editoimiseen täytyy käyttää automatisoidussa kuvan luomisessa sed-komentoa. Kirjautumistiedot annetaan kuvaa luodessa argumentteina. Avattu portti on kovakoodattu, koska SSH:n käyttämä vakioportti on 22. Docker-konteissa ei myöskään yleensä ole omaa Init-järjestelmää, kuten esimerkiksi systemd, koska kontit jakavat ytimen isäntäkoneen kanssa. Tämän takia OpenSSH-palvelin täytyy käynnistää sen omaa binääritiedostoa käyttäen, eikä systemctl-komennolla. Kuva toimii SSH-yhteydenoton testaamiseen, mutta se jakaa myös samat resurssit isäntäkoneen kanssa, joten kohdepalvelimet palauttavat identtiset järjestelmätiedot keskenään.

Tuotannossa kohdeympäristön ulkopuolelta ei kuitenkaan ole suoraa pääsyä palvelimille, vaan liikenteen täytyy kulkea välityspalvelimen kautta. Tämän simuloimiseksi kehitysympäristöön luodaan välityspalvelin käyttäen samaa kuvaa kuin kohdepalvelimiin, mutta sille asetetaan konfiguraatiossa sekä julkinen että yksityinen verkko. Näin tiedonkeruuohjelmisto

saa yhteyden välityspalvelimeen, josta välityspalvelin voi ottaa jatkoyhteyden yksityisessä verkossa oleviin kohdepalvelimiin. Kuvassa 20 on tiedonkerääjän, välityspalvelimen ja kohdekonttien konfiguraatiot.

```yaml
ssh-poller:
  build:
    context: ./
    dockerfile: Dockerfile
  container_name: ssh-poller
  volumes:
    - ./.env:/app/.env
  networks:
    - public_net
  ports:
    - $METRICS_PORT:$METRICS_PORT
  depends_on:
    - jump-box

▷Run Service
jump-box:
  build:
    context: ./test-target
    dockerfile: Dockerfile
    args: ···
  container_name: jump-box
  networks:
    - private_net
    - public_net
  ports:
    - "2222:22"
  depends_on:
    - target-server
  pid: "host"

▷Run Service
target-server:
  build:
    context: ./test-target
    dockerfile: Dockerfile
    args:
      - USER=${SSH_USER}
      - PW=${SSH_PASSWORD}
  networks:
    - private_net
  expose:
    - "22"
  pid: "host"
```

Kuva 20. Ote paikallisen kehitysympäristön Docker-compose konfiguraatiosta

Ympäristömuuttujat ja konttien rakennusargumentit haetaan pääasiallisesti projektin .env-tiedostosta. Tiedonkerääjä ohjelmoitiin kuitenkin käyttämään konfiguraatioonsa projektin juuritasolta löytyvää config.yml-tiedostoa (kuva 21).

```
! config.yml
1      # SSH-poller config
  1    metrics-port: 8080
  2
  3    jump-box:
  4      enabled: true
  5      address: jump-box
  6
  7    targets:
  8      - overwatch-target-server-1
  9      - overwatch-target-server-2
 10
 11
```

Kuva 21. SSH-kerääjän konfigurointitiedosto

Tiedonkerääjä tarvitsee toimintaansa varten kohdepalvelimien osoitteet, sekä portin mistä kerättyjen tietojen Prometheus-metriikat tarjotaan. Sen lisäksi konfiguraatiossa pitää määritellä täytyykö tiedonkerääjän mennä välityspalvelimen läpi. Tämä mahdollistaa, että monitorointijärjestelmää voidaan hyödyntää tarvittaessa muissakin ympäristöissä, joissa palvelimiin otetaan suoraan yhteys ilman välissä olevaa palvelinta. Konfigurointitiedostossa ei kuitenkaan säilytetä arkaluontoisempaa tietoa, kuten palvelimien käyttäjiä tai salasanoja.

6.1.2 Tiedonkerääjä

Tiedonkerääjällä on kolme pääasiallista toimintoa: Tiedon kerääminen, sen muuttaminen aikasarjamuotoon ja sen tarjoilu /metrics HTTP-päätepisteestä Prometheusta varten. Kuvassa 22 on tiedonkerääjän ajonaikaisia lokitulosteita.

```
ssh-poller       | 2025/05/18 11:10:59 Init complete
ssh-poller       | Config: {MetricsPort:8080 Proxy:{Enabled:true Address:jump-box} Targets:[overwatch-target-server-1 overwatch-targe
t-server-2]}
ssh-poller       | 2025/05/18 11:10:59 Starting ssh poller...
ssh-poller       | 2025/05/18 11:10:59 Attempting connection...
ssh-poller       | Accepted fingerprint for jump-box:22
ssh-poller       | 2025/05/18 11:10:59 Connection established
ssh-poller       | 2025/05/18 11:10:59 Polling target:  overwatch-target-server-1
ssh-poller       | 2025/05/18 11:10:59 Polling target:  overwatch-target-server-2
ssh-poller       | 2025/05/18 11:10:59 Prometheus exporter running on: :8080/metrics
```

Kuva 22. Tiedonkerääjän tulosteita

Ohjelma aloittaa ottamalla pysyvän SSH-yhteyden konfigurointitiedostossa määritellylle välityspalvelimelle, jonka jälkeen se aloittaa tiedonkeräämiseen kohdepalvelimilta. Ensimmäisessä paikallisessa versiossa ohjelman tiedonkeruuväli oli 10 sekuntia, jotta ohjelman toiminta saadaan todennettua nopeasti. Tiedonkeruuvälistäkin tehdään konfiguroitava asetus myöhemmässä käyttöönotossa. Kun yhteys välityspalvelimelle on muodostettu ja

tiedonkeruu on aloitettu, ohjelma avaa /metrics-päätepisteen jäsennellyn metriikkadatan tarjoilemista varten. Kuvassa 23 on /metrics-päätepisteen palauttamaa dataa.

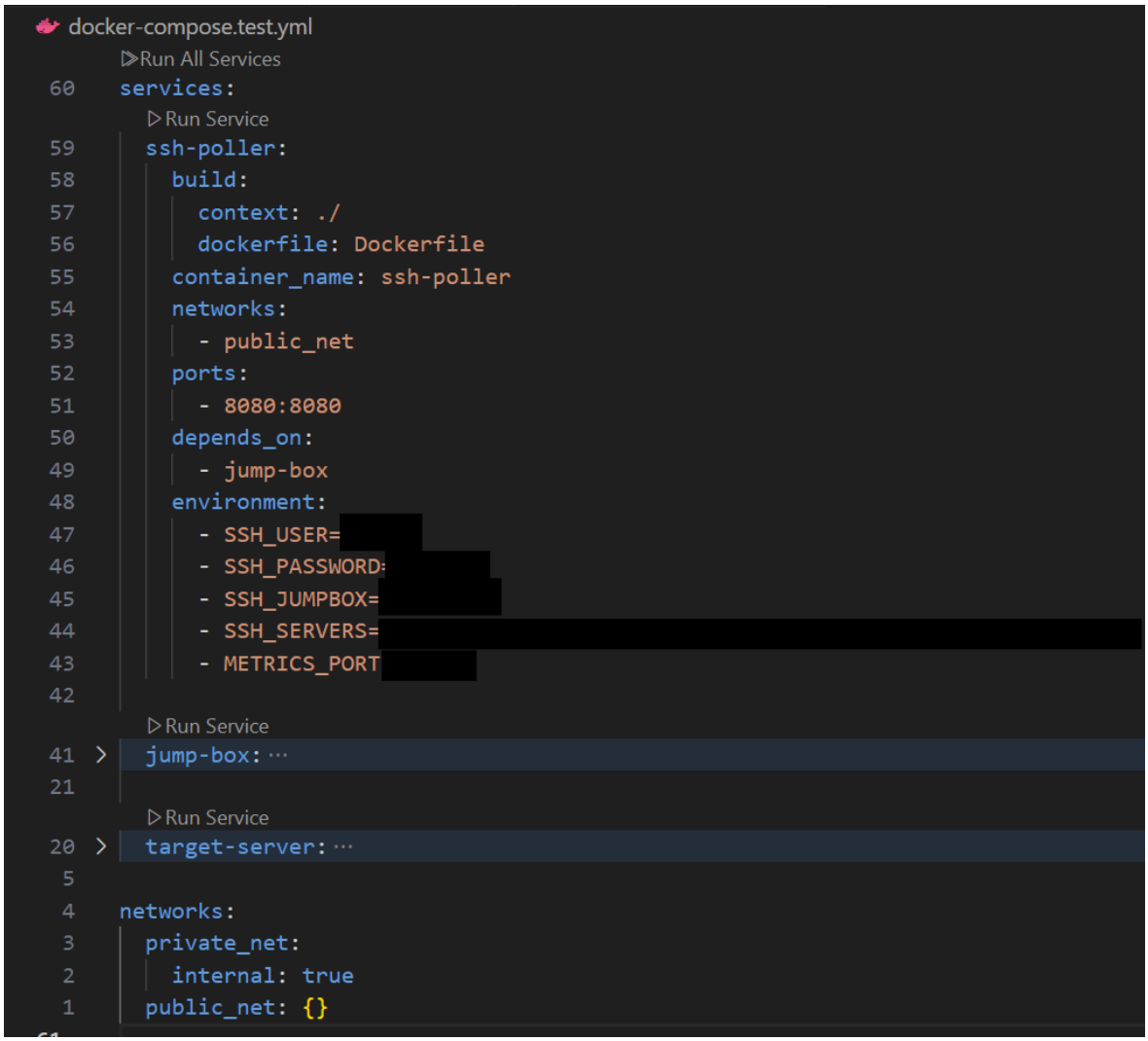
```
# HELP cpu_hardware_interrupts Generated by ssh-poller
# TYPE cpu_hardware_interrupts gauge
cpu_hardware_interrupts{instance="overwatch_target-server_1"} 0
cpu_hardware_interrupts{instance="overwatch_target-server_2"} 0
# HELP cpu_idle Generated by ssh-poller
# TYPE cpu_idle gauge
cpu_idle{instance="overwatch_target-server_1"} 99.4
cpu_idle{instance="overwatch_target-server_2"} 99.4
# HELP cpu_iowait Generated by ssh-poller
# TYPE cpu_iowait gauge
cpu_iowait{instance="overwatch_target-server_1"} 0.6
cpu_iowait{instance="overwatch_target-server_2"} 0.6
# HELP cpu_nice Generated by ssh-poller
# TYPE cpu_nice gauge
cpu_nice{instance="overwatch_target-server_1"} 0
cpu_nice{instance="overwatch_target-server_2"} 0
# HELP cpu_software_interrupts Generated by ssh-poller
# TYPE cpu_software_interrupts gauge
cpu_software_interrupts{instance="overwatch_target-server_1"} 0
cpu_software_interrupts{instance="overwatch_target-server_2"} 0
# HELP cpu_system Generated by ssh-poller
# TYPE cpu_system gauge
cpu_system{instance="overwatch_target-server_1"} 0
cpu_system{instance="overwatch_target-server_2"} 0
```

Kuva 23. Tiedonkerääjän /metrics-päätepisteen palauttamaa aikasarjadataa

Koska järjestelmää ajettiin paikallisessa konttipohjaisessa ympäristössä, jossa resurssit ovat jaettuja, kaikki kohdekontit palauttivat identtisiä isäntäkoneen metriikoita. Kehitysympäristössä voitiin kuitenkin todentaa, että ohjelman SSH-logiikka toimii perustasolla ja kerätyt metriikkatiedot jäsentyvät oikein.

6.1.3 Ohjelmistotestit

Järjestelmän toimivuuden varmistamiseksi kirjoitettiin kahdentyyppisiä ohjelmistotestejä: Yksikkötestit tiedonkerääjän jäsentelylogiikalle, sekä integraatiotesti, jolla varmistetaan järjestelmän yhteydenottologiikan toimivuus. Kuvassa 24 on integraatiotestiä varten tehtävän Docker-ympäristön konfigurointitiedosto.



Kuva 24. Testiympäristön docker-compose-konfiguraatiotiedosto

Integraatiotestissä rakennetaan testiversio paikallisesta ympäristöstä, josta on riisuttu pois Grafana ja Prometheus, koska ne ovat tarpeettomia tiedonkerääjän toiminnan testaamisen kannalta. Sen lisäksi ympäristömuuttujat kovakoodattiin poikkeuksellisesti konfiguraatiotiedostoihin, jotta testit voidaan ajaa ongelmitta Github runner-ympäristössä puskun yhteydessä ilman .env-tiedostoa.

Integraatiotesti tarkistaa, että tiedonkerääjä saa onnistuneesti yhteyden välityspalvelimeen ja kohdepalvelimiin, sekä palauttaa niiltä kerätyt tiedot onnistuneesti aikasarjamuodossa /metrics HTTP-päätepisteestä. Sen lisäksi se yrittää yhdistää konttiin, jota ei ole olemassa, varmistaakseen ettei epäonnistunut yhteydenotto vaikuta ohjelman muuhun toimintaan.

Yksikkötestejä varten on tallennettu tekstitiedostoihin top- ja cat-komentojen tulosteita eri tilanteista, kuten eri aikaa käynnissä olleista palvelimista tai viallisesti palautuneesta tulosteesta. Testit tarkistavat, että tulosteiden tiedot jäsentyvät virheettä jäsentelyfunktioissa, ja

tiedot täyttyvät tietorakenteisiin oikein. Alla on havainnollistava kuva testien onnistuneesta ajosta (Kuva 25).

```
~/overwatch/ go test -v ./...
2025/04/08 13:13:38 Init complete
=== RUN   TestIntegration
--- PASS: TestIntegration (41.58s)
=== RUN   TestParseOutput
--- PASS: TestParseOutput (0.00s)
=== RUN   TestParseOutputDays
--- PASS: TestParseOutputDays (0.00s)
=== RUN   TestParseOutputMalformed
--- PASS: TestParseOutputMalformed (0.00s)
=== RUN   TestFetchUptime
--- PASS: TestFetchUptime (0.00s)
PASS
```

Kuva 25. Testien ajaminen onnistuneesti.

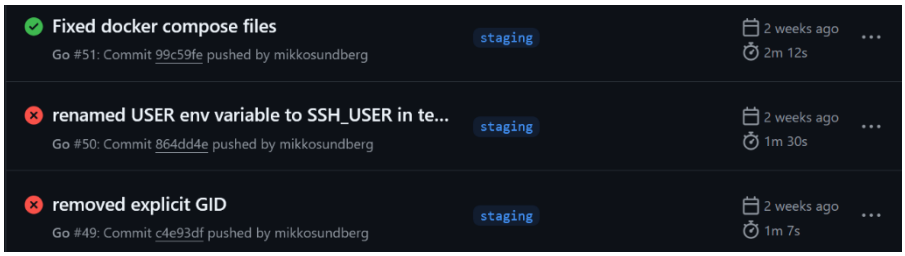
6.1.4 CI/CD-putki

Ohjelman kehityksen ja toimituksen helpottamiseksi, ohjelmiston testaus ja kontitus automatisoidaan Github-tietovarastossa (engl. repository) puskujen yhteydessä. Automatisoidulla testauksella voidaan tihentää päivitysten puskuväliä, koska Github Actions -tehtävät (engl. jobs) varmistavat, ettei uusi päivitys riko toiminnallisuutta, jos testikattavuus on riittävä. Automatisoinnilla helpotetaan myös ohjelmiston toimitusta Kubernetes-alustalle, koska se vaatii ohjelmiston kontituksen, sekä kontin tallentamisen yrityksen konttirekisteriin. Toteutuksessa noudatetaan myös yleisesti käytössä olevaa Development – Staging – Production GIT-haarakäytäntöä, jolla prosessi voidaan jakaa selkeisiin vaiheisiin helpottaen vaiheisiin liittyvää vianetsintää (GitLab 2025). Development-haaraan voidaan puskea raaka lähdekoodi, eikä sille ajeta Github Actions – tehtäviä. Kun lähdekoodi on valmis testattavaksi, se pusketaan Staging-haaraan, jossa sille ajetaan yksikkö- ja integraatiotestit. Kun ohjelmistotestit läpäistään, se voidaan liittää Production-haaraan ja lähettää tuotantoon.

```yaml
name: Go

on:
  push:
    branches: [ "staging" ]
  pull_request:
    branches: [ "staging" ]

jobs:

  build:
    runs-on: elisa-normal
    steps:
    - uses: actions/checkout@v4

    - name: Set up Go
      uses: actions/setup-go@v4
      with:
        go-version: '1.24.1'

    - name: Install dependencies
      run: go get .

    - name: Build
      run: mkdir bin && go build -v -o ./bin/ssh-poller

    - name: Clean up Docker environment
      run: |
        docker-compose -f docker-compose.test.yml down -v --remove-orphans
        docker system prune -af

    - name: Test
      run: go test -v ./...
```

Kuva 26. Github-tehtävä testien ajamiselle puskun yhteydessä

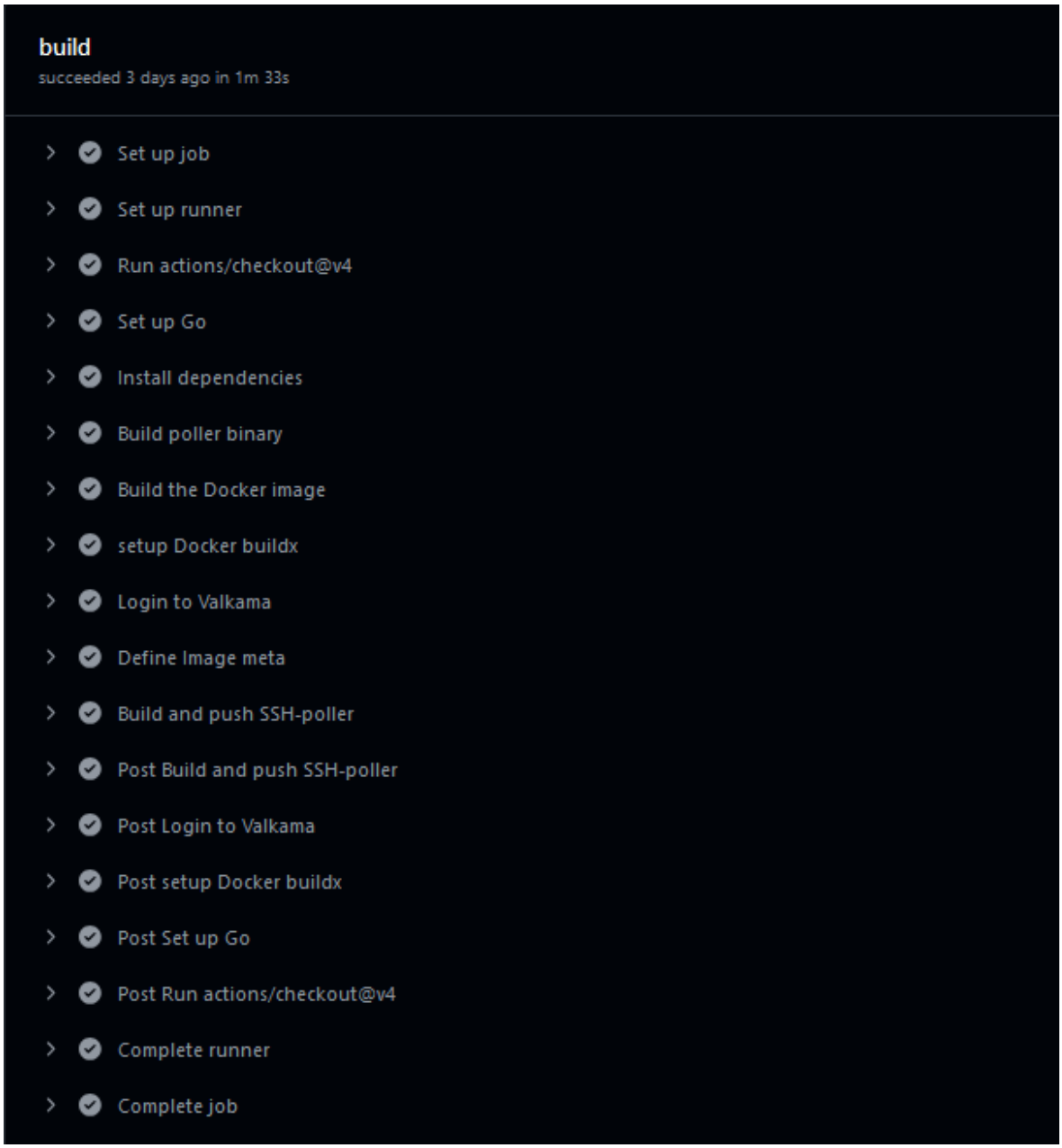
Testien ajamiseen puskun yhteydessä käytetään Github-ajoympäristöä (engl. Runner), jossa ennen testejä asennetaan Go ja tarvittavat riippuvuudet, rakennetaan binääritiedosto ja varmistetaan, että ajoympäristön Docker-ympäristössä ei ole jäänteitä edellisistä ajoker-roista. Tämän jälkeen testit ajetaan samalla komennolla kuin paikallisessa ympäristössä, ja ajoympäristö rakentaa testikontit integraatiotestausta varten. Kuvassa 27 havainnollistetaan Github Actions -tehtävien ajoja.



Kuva 27. Testien ajoja puskun yhteydessä

Ajon jälkeen ajoympäristö palauttaa joko onnistumisen tai epäonnistumisen. Ajon epäon-nistuminen ei kuitenkaan estä puskua tietovarastoon, mutta tämän tapahtuessa muutoksia ei pidä liittää päähaaraan ennen kuin testien ajo palauttaa onnistumisen.

Ensimmäinen vaihe järjestelmän käyttöönottoon on CI/CD-putken CD-vaiheen rakentaminen. Tässä projektissa se tarkoitti ohjelman kontitusta Docker-muotoon ja sen lähettämistä yrityksen sisäiseen konttirekisteriin, josta ohjelmiston ajamiseen käytettävä Kubernetes-cluster hakee kuvan. Kuvassa 28 on kuvaus CD-vaiheen Github Actions -tehtävän onnistuneesta ajosta.

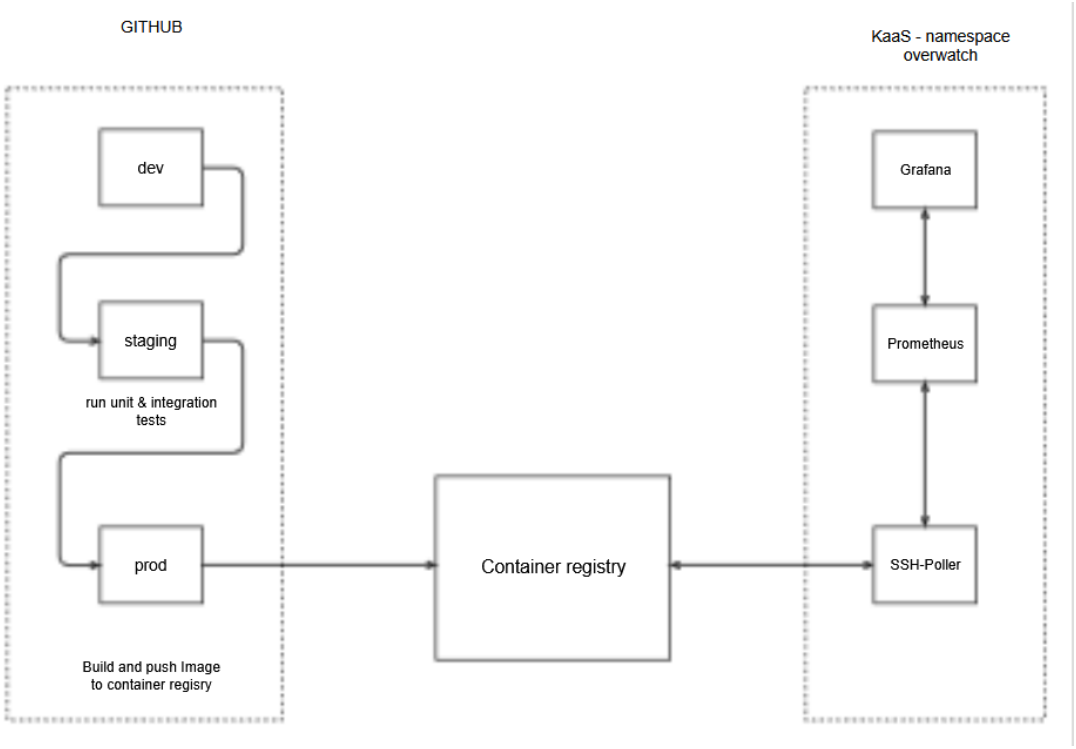


Kuva 28. Kontitus-vaiheen ajaminen Githubissa

Tehtävässä tiedonkerääjä käännetään ensin binääritiedostoksi, joka voidaan kopioida sen jälkeen rakennettavaan Docker-konttiin. Kontin rakentamisen jälkeen ajoympäristö autentikoi itsensä konttirekisteriin ja puskee käyttövalmiin kontin sinne.

6.2 Järjestelmän rakentaminen

Suunnittelun ja paikallisen kehitystyön jälkeen siirryttiin rakentamaan minimaalista versiota järjestelmästä toimeksiantajan KaaS-alustalle (Kubernetes-as-a-Service). Tämä tarkoittaa versiota, jossa järjestelmä kykenee ottamaan yhteyden monitoroitaviin kohteisiin ja kaikki järjestelmän monitorointiin tarvittavat palvelut on otettu alustalla käyttöön. Lisäksi niiden pitää pystyä kommunikoimaan keskenään järjestelmässä päästä päähän. Hälytyksiä, lopullisia näkymiä tai tapahtumien tallentamista tietokantaan ei tässä vaiheessa vielä kuitenkaan toteuteta. Salaisuuksien hallinta on huomioitu, mutta sitä ei työssä kuvata tarkemmin tietoturvallisuussyistä. Kuviossa 4 on ylätason kuvaus toteutettavasta kokonaisuudesta.



Kuvio 4. Kokonaiskuva järjestelmästä ja sen kehitysputkesta

Yllä olevassa kuviossa näkyvät aiemmassa kappaleessa toteutettu Github Actions – putki, toimeksiantajan konttirekisteri, sekä toimeksiantajan KaaS-alustalle (Kubernetes-as-a-Service) toteutettava ajoympäristö. KaaS-alustalla on Deployment-resurssit Grafanalle, Prometheukselle ja SSH-tiedonkerääjälle. Grafanan ja Prometheuksen konttikuvat haetaan toimeksiantajan ylläpitämästä konttirekisteristä, ja SSH-tiedonkerääjä aikaisemmin mainitusta konttirekisteristä, johon aikaisemmin toteutettu CI/CD-putki puskee kuvan.

6.2.1 Grafana

Grafanan käyttöönottoon käytettiin virallista Grafana Docker-konttia. Se riittää ainakin ensimmäiseen versioon toteutuksessa, jossa on tarkoitus todentaa järjestelmän toimivuus Kubernetes-ympäristössä. Käyttöönotto vaati deployment-resurssin Kubernetes-podien ajamista varten, Service-resurssin nimiavaruuden sisällä kommunikointia varten, sekä HTTPRoute-resurssin, jotta Grafanan käyttöliittymä on saatavilla toimeksiantajan sisäverkossa käyttöä varten. Kuvassa 29 on Grafana-deploymentin määrittelytiedosto.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  namespace: overwatch
  name: overwatch-grafana
  labels:
    app: overwatch-grafana
spec:
  replicas: 1
  selector:
    matchLabels:
      app: overwatch-grafana
  template:
    metadata:
      labels:
        app: overwatch-grafana
    spec:
      containers:
        - image: grafana/grafana:latest
          name: overwatch-grafana
```

Kuva 29. Deployment-konfigurointitiedosto Grafanalle

Deployment ei vaatinut tarkkoja määrittelyjä, koska käytössä on valmis ratkaisu. Valtaosa Grafanan konfigurointityöstä tapahtuu graafisen käyttöliittymän puolella, joten seuraavaksi voitiin siirtyä tekemään Service- ja HTTPRoute-resursseja, joilla Grafanan saa näkyviksi sekä käyttäjille, että muille nimiavaruudessa oleville deploymenteille. Kuvassa 30 on listattu käyttöönottoon tarvittavat resurssit.

```
mikkos@LPF3ZGNNM:~/overwatch-kaas$ ls -l grafana
total 12
-rw-r--r-- 1 mikkos mikkos 436 May  6 09:17 grafana-deployment.yaml
-rw-r--r-- 1 mikkos mikkos 551 May  6 09:51 ow-grafana-httproute.yaml
-rw-r--r-- 1 mikkos mikkos 233 May  6 09:28 ow-grafana-service.yaml
```

Kuva 30. Grafanan Kubernetes-käyttöönottoon tarvittavat konfigurointitiedostot

Jotta Grafanan käyttöliittymä saadaan tiimille näkyviin, sille täytyi määritellä reititys toimeksiantajan sisäverkkoon. Service- ja HTTPRoute-resurssien määrittelyjä ei kuvata työssä tarkasti.

6.2.2 Prometheus

Kuten Grafanan käyttöönotossa, myös Prometheukselle käytetiin valmista konttia toimeksiantajan konttirekisteristä. Prometheuksen toiminta vaatii kuitenkin myös oman konfigurointitiedostonsa, joka piti ottaa huomioon deployment-resurssin määrittelyssä. Kuvassa 31 on Prometheus-deploymentin määrittelytiedosto.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  namespace: overwatch
  name: overwatch-prometheus
  labels:
    app: overwatch-prometheus
spec:
  replicas: 1
  selector:
    matchLabels:
      app: overwatch-prometheus
  template:
    metadata:
      labels:
        app: overwatch-prometheus
    spec:
      containers:
        - image: prom/prometheus:latest
          name: overwatch-prometheus
          args:
            - "--config.file=/etc/prometheus/prometheus.yml"
            - "--storage.tsdb.path=/prometheus"
          volumeMounts:
            - name: ow-prom-config
              mountPath: /etc/prometheus/prometheus.yml
              subPath: prometheus.yml

            - name: prometheus-storage
              mountPath: /prometheus

      volumes:
        - name: ow-prom-config
          configMap:
            name: ow-prom-config
        - name: prometheus-storage
```

Kuva 31. Prometheus-deploymentin määrittelytiedosto

Prometheuksen tarvitsema konfiguraatiotiedosto prometheus.yml tallennettiin KaaS:iin ConfigMap-resurssina. Sen sisältöä ei työn raportissa kuvata tarkemmin, mutta siihen viitataan Prometheus-deploymentin määrittelyssä. ConfigMap kiinnitetään deploymentiin levyresurssina (engl. volume) ja Prometheus-kontille määritellään polku hakemistoon, josta konfiguraatio voidaan lukea. Kuvassa 32 on listaus Prometheuksen tarvitsemista konfigurointitiedostoista.

```
mikkos@LPF3ZGNNM:~/overwatch-kaas$ ls -l prometheus/
total 16
-rw-r--r-- 1 mikkos mikkos 284 May  6 11:19 ow-prom-configmap.yaml
-rw-r--r-- 1 mikkos mikkos 165 May  7 09:19 ow-prometheus-service.yaml
-rw-r--r-- 1 mikkos mikkos 914 May  7 09:12 prometheus-deployment.yaml
-rw-r--r-- 1 mikkos mikkos 144 May  6 11:13 prometheus.yml
```

Kuva 32. Prometheuksen Kubernetes-käyttöönottoon tarvittavat konfigurointitiedostot

Prometheukselle määritettiin service-resurssi, jotta se voi kommunikoida Grafanan sekä tiedonkerääjän kanssa. Prometheus tarjoaa myös graafisen käyttöliittymän käyttäjiä varten,

mutta sitä ei tässä tapauksessa asetettu saataville. Vain Grafanan on tarkoitus hakea sieltä dataa, joten toteutuksessa noudatetaan vähimpien oikeuksien periaatetta. Tarvittaessa käyttöliittymä voidaan kuitenkin asettaa saataville esimerkiksi ongelmanratkaisutilanteessa.

6.2.3 SSH-Poller

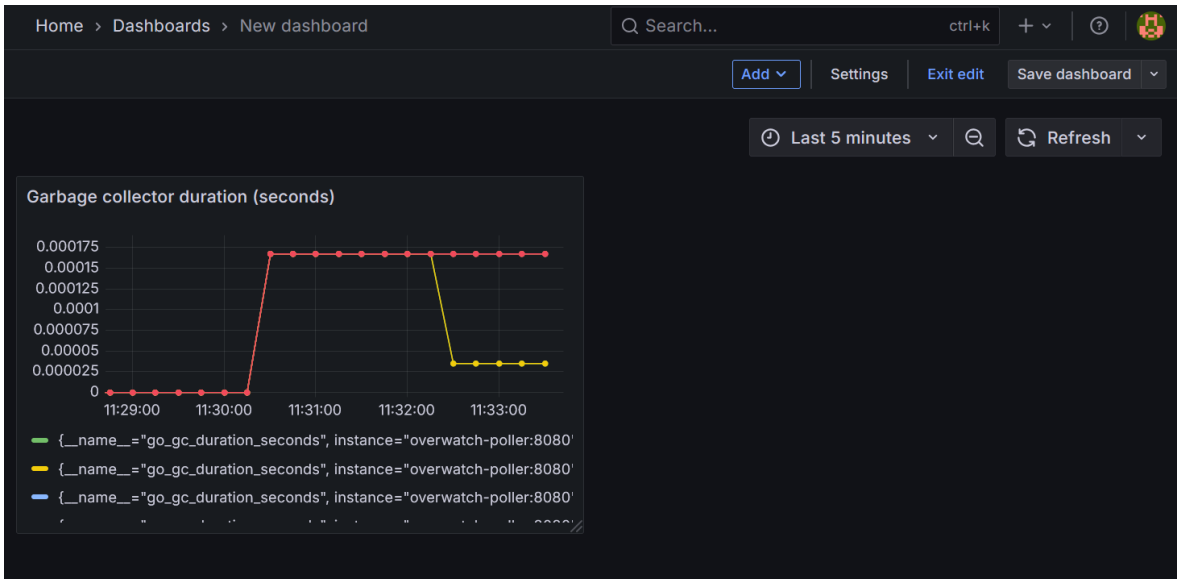
Viimeisenä käyttöönotossa aloitettiin rakentamaan SSH-pollerin eli tiedonkerääjän deploymenttia. SSH-tiedonkerääjän vaatimukset korkealla tasolla ovat melkein samat kuin Prometheuksella. Se tarvitsee deploymentin-resurssin ylläpitämään tiedonkerääjäpodia, sekä service-resurssin, jotta Prometheus saa siihen yhteyden metriikoiden keräämistä varten. Niiden lisäksi se tarvitsee ConfigMap-resurssin sen yleistä konfigurointia varten, sekä tarvittavat ympäristömuuttujat kontin ajamista varten. Näitä resursseja ei kuvata raportissa tarkasti.

Tiedonkerääjälle ei kuitenkaan rakennettu Kubernetes-ympäristöön testikohdetta, kuten paikallisessa ympäristössä. OpenSSH-palvelin vaatii toimintaansa root-käyttäjän oikeudet, jonka toimeksiantajan Kubernetes-alustan turvallisuuskäytännöt kieltävät. SSH-palvelimen konfiguroiminen toimimaan ilman root-oikeuksia olisi ollut liian työlästä testikohteen käyttöaikaan nähden, joten tiedonkerääjän kohteeksi asetettiin suoraan toimeksiantajan käytössä oleva palvelin. Tämä oli kuitenkin vasta yhteydenottotesti, jossa tiedonkerääjä käsitteli palvelinta välityspalvelimena. Se ei siis palauttanut vielä monitorointidataa. Kuvassa 33 on toteutetut palvelut toiminnassa.

```
mikkos@LPF3ZGNNM:~/overwatch-kaas$ kubectl get deployments
NAME                  READY   UP-TO-DATE   AVAILABLE   AGE
overwatch-grafana     1/1     1            1           11d
overwatch-poller      1/1     1            1           17d
overwatch-prometheus  1/1     1            1           10d
```

Kuva 33. Lopputuloksen esittely kubectl-työkalulla

Lopussa tiedonkerääjä sai yhteyden muodostettua kohdepalvelimelle. Lopputuloksena on dataputki, jossa metriikka kulkee onnistuneesti järjestelmän päästä päähän. Kuvassa 34 on kuva testinäkymästä, jossa Grafanalla on toteutettu kuvaaja tiedonkerääjän omasta metriikasta.



Kuva 34. Tiedonkerääjän roskienkeruun aktiivisuusaika Grafanassa

Dataputken toimivuus voitiin todentaa rakentamalla Grafanassa testinäkymä, joka käyttää datalähteenään järjestelmän Prometheus-instanssia. Koska Prometheus-instanssista voitiin hakea tiedonkerääjän metriikkaa, kykenevät järjestelmän osat kommunikoimaan keskenään. Tämä antoi pohjan tiedonkerääjän jatkokehitykselle, sekä erilaisten monitorointiratkaisujen toteuttamiselle.

7 Yhteenveto ja pohdinta

Lopullista järjestelmää ei opinnäytetyön laajuudessa ehditty rakentamaan, mutta kokonaisuudessaan toimeksiantaja oli pilotin toteutuksessa päästyyn vaiheeseen tyytyväinen. Järjestelmän toteutuksen ensisijainen tavoite oli rakentaa pohja monitorointijärjestelmälle, jonka päälle voidaan rakentaa erilaisia monitorointiratkaisuja. Toteutettu Grafana-Prometheus-putki tarjoaa monipuoliset mahdollisuudet kehittää järjestelmän toimintaa tarpeisiin sopivaksi. Työssä toteutetun SSH-tiedonkerääjän rinnalle voidaan esimerkiksi toteuttaa matalamman tason laitteistovalvontaan keskittyvä tiedonkerääjä, kuten Redfish Prometheus Exporter. Sen lisäksi Grafanaan voidaan liittää erilaisia tietokantoja, kuten SQL-tietokanta hälytysten tallentamiseen, tai aikasarjatietokanta metriikkadatan pitkäaikaiseen säilytykseen. Lisäksi tiedonkerääjän onnistunut toiminta paikallisessa ympäristössä loi uskoa SSH-pohjaisen ratkaisun mahdollisuuksiin.

SSH-pohjaista metriikankeruuta voidaan myös jatkokehittää. Pilotissa se ottaa yhteyden SSH-palvelimelle, ajaa määritellyt komennot, jäsentelee komentojen tulosteen sille tarkoitettuun tietorakenteeseen ja tarjoilee sen Prometheus-metriikkamuodossa kerättäväksi. Uusien ominaisuuksien kehittäminen on siis melko suoraviivaista. Lisätään uuden komennon ajofunktio, sekä komennon tulosteen tietorakenne ja jäsentelyfunktio. Tämä voi olla joissakin tilanteissa työlästä, mutta ratkaisun vahvuus on sen monikäyttöisyydessä. Koska tiedonkeräyslogiikka tapahtuu palvelimen ulkopuolella, täytyy palvelimilta löytyä vain Linux vakiotyökalut, jotka pääasiallisesti löytyvät jokaiselta Linux-palvelimelta.

SSH-pohjaisen ratkaisun heikkous oli käyttöönottoon ja suunnitteluun tarvittava työ- ja aikamäärä. Sen lisäksi kolmannen osapuolen sovellukset tarjoavat päivityksiä ohjelmistovirheiden korjaamiseksi, mutta itse kirjoitetun tiedonkeräyslogiikan ylläpito jää kokonaan tiimin vastuulle. Vaikka jatkokehitys on yksinkertaista, täytyy jokaisen uuden komennon jäsentelylogiikka kehittää alusta asti itse. Jos järjestelmä rakennetaan ympäristöön, jossa monitoroitaville palvelimille voidaan asentaa kolmannen osapuolen agenttiohjelmisto, voi valmiin ratkaisun käyttö olla ylläpidon ja luotettavuuden kannalta perusteltua.

Ensimmäisen version jälkeen järjestelmään tullaan lisäämään tarvittavia ominaisuuksia, kuten esimerkiksi konfiguroidut Grafana-hälytykset ja -varoitukset, tietokantayhteys sekä valmis pohja uusien Grafana-näkymien luomiseksi, jotta uusien palvelimien lisääminen järjestelmän kohteiksi olisi saumattomampaa.

Lähteet

Buchanan, I 2025. Containers vs. virtual machines. Viitattu 18.5.2025

Saatavissa https://www.atlassian.com/microservices/cloud-computing/containers-vs-vms

Burns, B. & Grant, B. & Oppenheimer, D. & Brewer, E. & Wilkes, J. 2016. Borg, Omega, and Kubernetes, ACM Queue Vol.14(1). Viitattu 8.5.2025

Saatavissa https://queue.acm.org/detail.cfm?id=2898444

Docker. 2025a. Docker compose – Getting started. Viitattu 17.4.2025)

Saatavissa https://docs.docker.com/compose/gettingstarted/

Docker. 2025b. What is a container? Viitattu 16.4.2025

Saatavissa https://www.docker.com/resources/what-container/

Elisa. 2025. Elisa lyhyesti, Viitattu 12.5.2025

Saatavissa https://elisa.fi/yhtiotieto/

Gitlab. 2025. What are Git version control best practices? Viitattu 9.4.2025

Saatavissa https://about.gitlab.com/topics/version-control/version-control-best-practices/

GNU. 2025. Sed, a stream editor. Viitattu 12.5.2025

Saatavissa https://www.gnu.org/software/sed/manual/sed.html

Grafana. 2025. Overview. Viitattu 18.5.2025

Saatavissa https://grafana.com/docs/grafana/latest/

Kubernetes. 2025a. Cluster Architecture. Viitattu 11.5.2025

Saatavissa https://kubernetes.io/docs/concepts/architecture/

Kubernetes. 2025b. Deployment, Viitattu 9.5.2025

Saatavissa https://kubernetes.io/docs/concepts/workloads/controllers/deployment/

Kubernetes. 2025c. Ingress. Viitattu 9.5.2025

Saatavissa https://kubernetes.io/docs/concepts/services-networking/ingress/

Kubernetes. 2025d. Overview. Viitattu 8.5.2025

Saatavissa https://kubernetes.io/docs/concepts/overview/

Kubernetes. 2025e. Pods. Viitattu 8.5.2025

Saatavissa https://kubernetes.io/docs/concepts/workloads/pods/

Kubernetes. 2025f. Service, Viitattu 9.5.2025

Saatavissa https://kubernetes.io/docs/concepts/services-networking/service/

man7.org. 2025a. awk(1p) — Linux manual page Viitattu 1.5.2025

Saatavissa https://man7.org/linux/man-pages/man1/awk.1p.html

man7.org. 2025b. cat(1) — Linux manual page. Viitattu 30.4.2025

Saatavissa https://man7.org/linux/man-pages/man1/cat.1.html

man7.org. 2019a. grep(1) — Linux manual page. Viitattu 30.4.2025

Saatavissa https://man7.org/linux/man-pages/man1/grep.1.html

man7.org. 2019b. proc(5) — Linux manual page. Viitattu 29.4.2025

Saatavissa https://man7.org/linux/man-pages/man5/proc.5.html

man7.org. 2017. sed(1p) — Linux manual page. Viitattu 1.5.2025

Saatavissa https://man7.org/linux/man-pages/man1/awk.1p.html

man7.org. 2025c. tail(1) — Linux manual page. Viitattu 30.4.2025

Saatavissa https://man7.org/linux/man-pages/man1/tail.1.html

man7.org. 2025d. top(1) – Linux manual page. Viitattu 29.4.2025

Saatavissa https://man7.org/linux/man-pages/man1/top.1.html

Prometheus. 2025a. Data Model. Viitattu 5.5.2025

Saatavissa https://prometheus.io/docs/concepts/data_model/

Prometheus. 2025b. Metric types, Viitattu 5.5.2025

Saatavissa https://prometheus.io/docs/concepts/metric_types/

Red Hat . 2025. What is CI/CD? Viitattu 7.4.2025. (malli)

Saatavissa https://www.redhat.com/en/topics/devops/what-is-ci-cd

Sharma, S. 2022. Top vs htop: What’s the Difference? Viitattu 18.5.2025

Saatavissa https://linuxhandbook.com/top-vs-htop/

StackOverflow. 2024. StackOverflow developer survey, Technology. Viitattu 18.5.2025

Saatavissa https://survey.stackoverflow.co/2024/technology

Thomas, T. 2018. macOS configuration with GNU Programs. Viitattu 18.5.2025

Saatavissa https://medium.com/@todd.dsm/macos-configuration-with-gnu-programs-373fbc131f7f