Tampereen ammattikorkeakoulu

Automaattinen datan keräys ja raportointi

Petri Pellinen

OPINNÄYTETYÖ
Joulukuu 2025

Tietotekniikka
Ohjelmistotekniikka

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietotekniikan tutkinto-ohjelma
Ohjelmistotekniikka

PELLINEN, PETRI:
Automaattinen datan keräys ja raportointi

Opinnäytetyö 52 sivua, joista liitteitä 11 sivua
Joulukuu 2025

Opinnäytetyön tavoitteena oli kehittää automatisoitu prosessi VMware-virtuaaliympäristön datan keräämiseksi, käsittelemiseksi ja tallentamiseksi Google Driveen. Ratkaisu toteutettiin Python-ohjelmointikieltä hyödyntäen ja ajettiin Linux-palvelimella, jossa ajoituksista huolehti crontab. Ratkaisu heijastaa alan viimeaikaisia kehitystrendejä, joissa automaatio, avoimen lähdekoodin työkalut sekä pilvipalvelujen hyödyntäminen ovat keskeisessä asemassa nykyaikaisessa IT-hallinnassa.

Projektin aikana painottuivat tietoturva, ylläpito ja järjestelmän skaalautuvuus. Skripti kehitettiin siten, että se reagoi sekä järjestelmä- että API-päivityksiin, ja kaikki tuntemattomat virhe- ja poikkeustilanteet käsitellään robustisti informatiivisilla virheilmoituksilla. Ylläpitoa helpottivat selkeä dokumentaatio ja sisäiset ohjeet, jotka mahdollistavat kokonaisuuden hallinnan ja vastuun siirron tulevaisuudessa. Tulosten perusteella järjestelmä helpotti ylläpidon päivittäistä työtä erityisesti vikatilanteiden selvityksessä, resurssisuunnittelussa ja kapasiteetin hallinnassa. Automaattisesti päivittyvä Google Sheets-taulukko antoi ylläpidolle mahdollisuuden tarkastella ja etsiä virtuaalikoneiden avaintietoja, kuten host-palvelimia, resursseja ja tallennuskapasiteettia, nopeasti eri käyttötilanteissa. Järjestelmän tehokkuus, virheensietokyky ja matalat laitteistovaatimukset osoittivat, että ratkaisu on helposti monistettavissa ja hyödynnettävissä erilaisissa organisaatioissa.

Opinnäytetyön myötä syvenivät tekijän tekniset taidot, projektinhallintakokemus ja ymmärrys IT-infrastruktuurien automatisoinnin merkityksestä. Työn keskeinen innovaatio on automaattinen Python-skripti, joka kerää tietoja vSphere API:n kautta, käsittelee datan taulukkomuotoon ja siirtää sen Google Driveen Google API-kirjastojen avulla. Ratkaisu mahdollistaa ajantasaisen ja keskitetysti saavutettavan tiedon hyödyntämisen ilman manuaalisia työvaiheita. Lopputulos osoitti, että hyvin suunniteltu automaatio tuottaa merkittävää lisäarvoa niin käytännön ylläpidolle kuin koko organisaation toimintavarmuudelle ja resurssien käytölle.

Asiasanat: automaatio, ylläpito, virtuaalikone, ohjelmointirajapinnat

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Information Technology
Software Engineering

Petri Pellinen:
Automatic Data Collection and Reporting

Bachelor's thesis 52 pages, appendices 11 pages
December 2025

The objective of this thesis was to develop an automated process for collecting, processing, and storing VMware virtual environment data in Google Drive. The solution was implemented using Python programming language and executed on a Linux server, with scheduling handled by crontab. The core innovation of the work is an automated Python script, which collects data through the vSphere API, processes it into tabular form, and transfers it to Google Drive via Google API libraries. This solution enables the use of up-to-date, centrally accessible information without the need for manual intervention.

The project emphasized information security, maintainability, and system scalability. The script was developed to adapt to both system and API updates, and all unexpected error and exception scenarios are robustly handled with informative error messages. Maintenance is supported by clear documentation and internal guidelines, which enable comprehensive management and efficient transfer of responsibilities in the future. According to the results, the system significantly facilitated daily operational work, especially in troubleshooting, resource management, and capacity planning. The automatically updated Google Sheets table allowed administrators to quickly review and search for key virtual machine information such as host servers, resources, and storage capacity in various operational scenarios. The efficiency, fault tolerance, and low hardware requirements of the system demonstrated that the solution can be easily replicated and utilised in different organizations.

Throughout the project, the author’s technical skills, project management experience, and understanding of the importance of IT infrastructure automation were deepened. The solution reflects recent industry trends where automation, open-source tools, and the utilisation of cloud services play a central role in modern IT management. The outcome demonstrated that well-planned automation provides significant added value not only for daily maintenance but also for the overall operational reliability and resource utilization of the organisation.

Key words: automation, maintenance, virtual machines, APIs

SISÄLLYS

ERITYISSANASTO

API (Application Programming Interface)	Ohjelmointirajapinta
VM (Virtual Machine)	Virtuaalikone
Host	Isäntäpalvelin
Python	Ohjelmointikieli
Skripti	Suoritettava ohjelmisto

1 JOHDANTO

Virtuaalikoneiden (VM, virtual machine) hallinta on keskeinen osa organisaatioiden nykyaikaista IT-infrastruktuuria. Virtuaaliympäristöjen käyttöönotto mahdollistaa resurssien tehokkaan ja joustavan hallinnan sekä tukee skaalautuvia ja kustannustehokkaita ratkaisuja (Google Cloud, 2024). Virtuaaliympäristöjen kasvava laajuus ja monipuolisuus ovat kuitenkin lisänneet tarvetta entistä kattavampaan ja järjestelmällisempään järjestelmäseurantaan. Tietojen kerääminen, käsittely ja raportointi virtuaalikoneympäristöistä ovat usein manuaalisia ja aikaa vieviä tehtäviä, mikä altistaa prosessin mahdollisille virheille sekä kuormittaa ylläpitäjiä. IT-infrastruktuurin automatisointi onkin noussut keskeiseksi keinoksi helpottaa hallinnointia, parantaa tiedon laatua ja vapauttaa asiantuntijoiden työaikaa muihin tehtäviin. Lisäksi pilvipalvelujen, kuten Google Driven, hyödyntäminen tukee tiedon keskitettyä tallentamista ja jakamista eri sidosryhmille.

Tämän opinnäytetyön tavoitteena on suunnitella ja toteuttaa järjestelmä, joka automatisoi VMware-virtuaaliympäristön tietojen keräämisen, prosessoinnin ja tallennuksen Google Drive -pilvipalveluun. Tarkoituksena on kehittää Python-ohjelmointikieleen perustuva järjestelmä, joka mahdollistaa tiedonkeruun, käsittelyn ja raportoinnin automatisoidusti. Työ on rajattu koskemaan VMware-ympäristöä ja Google Drive-integraatiota, eikä siihen sisälly muiden virtualisointialustojen tarkastelua tai laajojen yritysverkkojen kokonaisvalvontaa.

Projektin kohderyhmänä ovat erityisesti IT-infrastruktuurista ja järjestelmien ylläpidosta vastaavat asiantuntijat ja sidosryhmät, jotka tarvitsevat ajantasaista ja luotettavaa tietoa virtuaalikoneympäristön tilasta. Ratkaisun tavoitteena on tarjota tehokas ja helppokäyttöinen tapa automatisoida VM-datan keruu sekä parantaa tiedon saavutettavuutta.

2 PROJEKTIKUVAUS

Tässä luvussa käydään läpi projektin tarkoitus. Projektin ongelman määrittelyssä kuvataan sitä, miksi projekti on ylipäätään tarpeellista toteuttaa. Toiseksi kuvataan ongelmaan suunniteltu ratkaisu, joka sopii käytössä oleviin järjestelmiin.

2.1 Projektin ongelman määrittely

Virtuaalikoneympäristöjen määrä on kasvanut huomattavasti organisaatioiden tietojärjestelmien kehityksen myötä. Yhä useammin palvelut, sovellukset ja datan varastointi toteutetaan virtuaalikoneiden avulla, koska ratkaisut ovat skaalautuvia, joustavia ja kustannustehokkaita (IBM, 2024). Tämä kehitys on kuitenkin lisännyt myös järjestelmäylläpitäjien vastuuta valvoa ja hallinnoida laajoja virtuaaliympäristöjä tehokkaasti (IBM, 2024).

Perinteisesti virtuaalikoneiden tilatietojen, kuten suorittimen käytön, muistinkäytön ja levytilan, keruu sekä raportointi ovat olleet manuaalisia prosesseja. Lisäksi ylläpidolle on olennaista saada kootusti haltuun virtuaalikoneiden verkkotiedot. Manuaalinen tiedonkeruu on kuitenkin aikaa vievää, altista virheille ja hankaloittaa tiedon ajantasaista jakamista. Myös datan indeksointi, säilytys ja jakelu pilvipalveluihin saattaa vaatia useita työvaiheita ja erityisosaamista käyttäjiltä.
Haasteena on erityisesti järjestelmän tehokas seuranta ja datan saattaminen helposti saataville riippumatta siitä, missä käyttäjät tai sidosryhmät sijaitsevat. Tiedon hajanaisuus hidastavaa prosessia ja voi aiheuttaa riskejä esimerkiksi IT-infrastruktuurin valvonnassa.

Tässä projektissa pyritään ratkaisemaan erityisesti seuraavat ongelmat:

- Virtualisaatioympäristön tilatietojen manuaalisen keruun ja seurannan kuormittavuus
- Virheiden mahdollisuus tiedon monivaiheisessa käsittelyssä
- Tiedon jakamisen ja tallentamisen hajanaisuus eri järjestelmien välillä
- Tarve tehdä datasta helposti saavutettavaa ja automaattisesti päivittyvää koko organisaatiolle

Yhteenvetona voidaan todeta, että automatisoitujen työnkulkujen puute vaikeuttaa virtuaalikoneiden tehokasta seurantaa ja lisää manuaaliseen työhön liittyvää

virheriskiä sekä hidastaa tiedon jakamista. Tämä luo tarpeen ratkaisulle, joka poistaa manuaalisia työvaiheita, keskittää tiedon ja mahdollistaa virtuaalikoneiden ajantasaisen seurannan ja raportoinnin.

2.2 Ehdotettu ratkaisu

Ongelman ratkaisuksi tässä projektissa toteutetaan automatisoitu järjestelmä, jonka ydin muodostuu Python-ohjelmointikielellä laaditusta skriptistä. Ratkaisussa hyödynnetään virtuaaliympäristöä, jossa skripti suoritetaan aikataulutetusti Linux-järjestelmän Crontab-työkalun avulla. Näin varmistetaan toistuva ja säännöllinen tiedonkeruu ilman manuaalista käynnistämistä. Python-skripti hyödyntää VMware-ympäristön ohjelmointirajapintaa (API), jonka avulla skripti kerää olennaiset tiedot virtuaalikoneista. Näihin tietoihin kuuluvat esimerkiksi suorittimen ja muistin käyttö sekä levytila. Skripti toimii itsenäisesti ja hakee datan suoraan VMware-palvelimelta, jolloin vältytään manuaalisilta työvaiheilta ja inhimillisiltä virheiltä.

Kerätty data käsitellään skriptissä siten, että tieto muotoillaan helppokäyttöiseen ja luettavaan taulukkomuotoon. Datan käsittelyssä käytetään tarkoituksenmukaisia Python-kirjastoja, joiden avulla se voidaan tallentaa Google Sheets-yhteensopivaksi tiedostoksi. Lopuksi skripti hyödyntää Google Drive-sovellusrajapintaa (API), jonka avulla käsitelty tiedosto tallennetaan automaattisesti keskitetysti Google Driveen. Näin tuorein data on saatavilla ja käytettävissä ajon ajastuksesta riippuen.

Toteutettu ratkaisu muodostaa automaatioketjun: virtuaalikoneiden tilatiedot kerätään, muokataan haluttuun muotoon ja ladataan ajantasaisesti turvalliseen pilvitallennukseen ilman käyttäjän erillistä panosta. Tämä parantaa datan saavutettavuutta, saa aikaan ajansäästöä sekä vähentää virheiden mahdollisuutta koko prosessin ajan.

3 TYÖKALUT, TEKNOLOGIAT JA JÄRJESTELMÄARKKITEHTUURI

Tässä luvussa käydään läpi projektissa käytettyjä työkaluja ja teknologioita. Eri teknologioiden ja työkalujen hyödyt ja käyttötarkoitukset projektin kannalta. Luvussa kuvataan projektin osat ja niiden rooli projektin kokonaisuudessa.

3.1 Python

Python-ohjelmointikieli valittiin tämän projektin toteutuksen perustaksi sen joustavuuden, laajan yhteisön tuen sekä laajasti saatavilla olevien avoimen lähdekoodin kirjastojen vuoksi. Python soveltuu erityisen hyvin automaatiosovelluksiin ja tiedonkeruuseen, sillä sen syntaksi on selkeä ja helposti ylläpidettävä (Python Software Foundation, 2024). Lisäksi Python tukee laajasti erilaisia ohjelmointirajapintoja (API), joiden hyödyntäminen on keskeistä tämänkaltaisissa integraatiopainotteisissa projekteissa (Python Software Foundation, 2024).
Pythonin eduiksi voidaan myös lukea alustariippumattomuus ja hyvä yhteensopivuus sekä Linux- että Windows-ympäristöjen kanssa. Python-skriptien avulla voidaan helposti automatisoida toistuvia tehtäviä ja käsitellä monimutkaisiakin datarakenteita tehokkaasti (DataCamp, 2024).

Projektin toteutuksessa käytettiin useita keskeisiä kirjastoja ja kehysratkaisuja, jotka nopeuttivat ohjelmointia ja paransivat sovelluksen luotettavuutta:
pyvmomi: VMware-ympäristöjen ohjelmointirajapinta (Python VMware vSphere API), jota hyödynnettiin virtuaalikoneiden tietojen keräämisessä.
google-auth ja google-api-python-client: Google Driven API-rajapinnan hyödyntämisessä olennaiset kirjastot, joiden avulla toteutettiin tiedostojen autentikointi, valtuutus sekä saumaton tiedonsiirto Google Drive -pilvipalveluun.
Näiden kirjastojen avulla kokonaisuus voitiin rakentaa modulaarisesti ja luotettavasti, mikä mahdollisti tehokkaan ja laadukkaan automatisointiratkaisun kehittämisen. Pythonin avulla oli myös helppo testata, ylläpitää ja jatkokehittää ratkaisua muuttuvien tarpeiden mukaan.

3.2 VMware

Virtuaalikoneympäristön tietojen tehokas ja luotettava kerääminen edellyttää suorien ohjelmointirajapintojen hyödyntämistä. VMware tarjoaa kattavan API-rajapin-

nan (Application Programming Interface), jonka avulla voidaan ohjelmallisesti hakea tietoja virtuaalikoneista sekä hallita virtuaaliympäristöä ilman tarvetta manuaaliin käyttöliittymässä työskentelyyn (Broadcom, 2024).

Tässä projektissa VMware-ympäristön tietojen haku toteutettiin käyttämällä pyvmomi-kirjastoa, joka on Python-kielinen asiakasohjelmakirjasto VMware vSphere API:lle. Pyvmomi mahdollistaa yhteyden muodostamisen vSphere-palvelimeen, virtuaalikoneiden selaamisen sekä yksityiskohtaisten tilatietojen, kuten suorittimen käyttötason, muistin käytön ja levytilan, noutamisen ohjelmallisesti. Kirjasto on avoimen lähdekoodin ja sitä tuetaan aktiivisesti yhteisön toimesta, mikä mahdollistaa luotettavan integraation VMware-järjestelmien kanssa.
Pyvmomi-kirjastossa autentikointi tapahtuu VMware vCenterin kautta, minkä jälkeen skripti pystyy ohjelmallisesti hakemaan tiedot halutuista virtuaalikoneista. Näitä tietoja voidaan myös suodattaa ja käsitellä edelleen projektin tarpeiden mukaisesti.

3.3 Google Drive

Google Drive API:lla oli keskeinen rooli projektissa tiedon tallennuksen ja jakelun automatisoinnissa. API:n (Application Programming Interface) avulla projektissa kehitetty Python-skripti pystyy kommunikoimaan Google Driven kanssa ohjelmallisesti, jolloin kerätyt ja käsitellyt tiedot voidaan siirtää suoraan pilvipalveluun ilman manuaalista välivaihetta (Google Developers Drive, 2024). Tämä mahdollistaa kerätyn datan keskitetyn, ajantasaisen ja helposti saavutettavan säilyttämisen, jolloin sidosryhmät ja käyttäjät voivat tarkastella tuloksia halutessaan eri sijainneista ja laitteista.

Jotta automatisointi olisi turvallista ja käyttäjätietoja suojeltaisiin, Google Drive API:n käyttöön liittyy todennus- ja valtuutusprosessi. Skripti käyttää Google Auth-kirjastoa, jonka avulla käyttäjä tunnistetaan (OAuth 2.0 -protokolla) (Google Developers OAuth, 2024). Ensimmäisellä käyttökerralla käyttäjä myöntää tarvittavat oikeudet skriptille, jonka jälkeen voidaan hyödyntää päteviä token-tiedostoja seuraavia kirjautumiskertoja varten. Tämän valtuutusprosessin avulla varmistetaan, että vain oikeudet omaava ohjelma voi ladata ja päivittää tiedostoja käyttäjän Google Drive -tilille.

Integraation avulla skripti voi:

- Ladata luodun taulukon (esim. CSV- tai Excel-tiedoston) suoraan määrättyyn kansioon Google Drivessa
- Luoda ja päivittää tiedostoja automaattisesti ilman käyttäjän käsin tekemiä toimenpiteitä
- Hallinnoida tiedostojen jakamista ja käyttöoikeuksia Drive-ympäristössä

Sähköisen valtuutuksen ja pyyntöjen autentikoinnin ansiosta tiedonsiirto Driveen on sekä turvallista että automaation näkökulmasta luotettavaa.

3.4 Linux-palvelin ja crontab

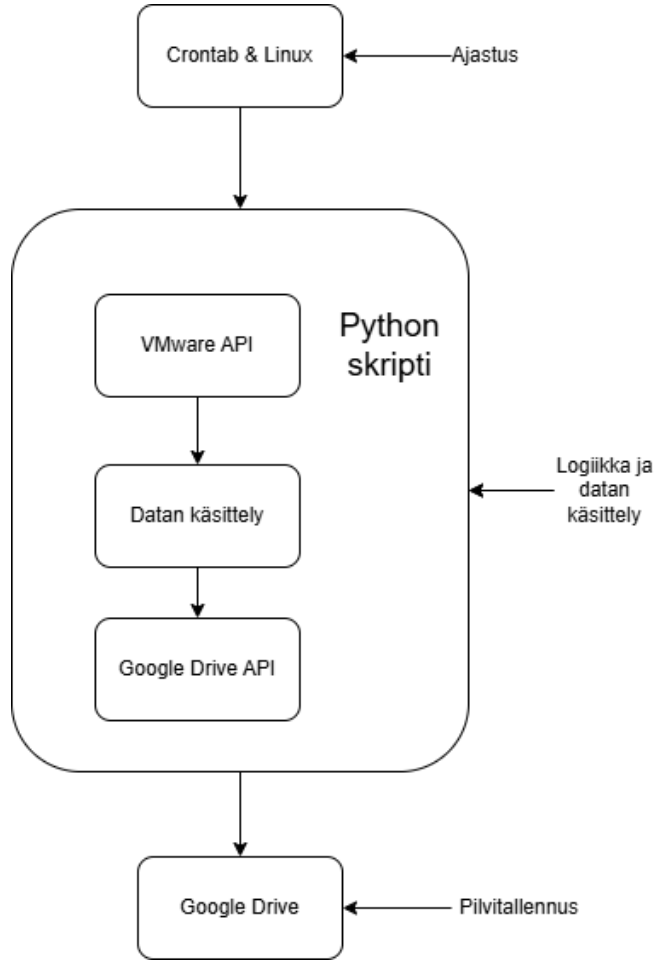
Projektissa käyttöympäristöksi valittiin Linux-palvelin, joka tarjoaa useita etuja automaatiotehtävien toteutuksessa. Linux on tunnettu vakaudestaan, joustavuudestaan ja skaalautuvuudestaan palvelinkäytössä, minkä lisäksi se on avoimen lähdekoodin järjestelmä ja laajalti tuettu yritysympäristöissä. Linux-ympäristössä on runsaasti valmiita työkaluja ja komentorivityökaluja, jotka helpottavat automaattisten prosessien toteuttamista ja ylläpitoa. Kehittynyt paketinhallinta, skriptauksen mahdollisuudet ja tehokas resurssien hallinta tekevät Linuxista erinomaisen alustan myös Python-pohjaisille automatisointiratkaisuille.

Osana projektin automaatiota hyödynnettiin Linuxin sisäänrakennettua ajastusohjelmaa, crontabia. Crontab (cron table) mahdollistaa aikataulutettujen tehtävien ajon automaattisesti määritetyin välein – esimerkiksi tunneittain, päivittäin tai viikoittain. Crontabiin luotu ajoitusmerkintä varmistaa, että Python-skripti suoritetaan toistuvasti ilman käyttäjän erillistä toimintaa. Tässä käyttötarkoituksessa skriptin on ajastettu viikon välein. Näin pystytään automatisoimaan virtuaalikoneympäristön tiedonkeruu sekä tiedon jatkokäsittely ja -siirto Google Driveen täysin itsenäisesti.

Crontabin käyttö vähentää manuaalisen työn tarvetta, pienentää unohdusten ja inhimillisten virheiden riskiä sekä parantaa järjestelmän luotettavuutta ja tiedon ajantasaisuutta. Linux-palvelin ja crontab yhdessä mahdollistavat automatisoidun työnkulun, jossa datan keruu, käsittely ja tallennus tapahtuvat säännöllisesti ja ennustettavasti.

3.5 Arkkitehtuurikaavio

Kuvassa näkyy visuaalinen esitys projektin arkkitehtuurista. Seuraavissa kappaleissa käsitellään projektin osia tarkemmin yksitellen.



KUVA 1. Järjestelmän arkkitehtuurikuva

3.6 Työnkulun kuvaus

Järjestelmän työnkulku rakentuu useasta selkeästä ja toisiaan täydentävästä vaiheesta, joiden tarkoituksena on automatisoida virtuaalikoneympäristön tilatietojen keruu, käsitellä nämä tiedot haluttuun muotoon ja tallentaa lopputulos keskitetysti Google Driveen. Työnkulku on suunniteltu siten, että se toimii täysin ilman käyttäjän manuaalista ohjausta Crontabin ajastuksen ansiosta.

3.6.1 Vaihe 1: Ajastus ja käynnistys

Työ alkaa, kun Linux-palvelimen Crontab-työkalu ajoittaa ja käynnistää Python-skriptin etukäteen määritettynä ajankohtana kerran viikossa. Tämä takaa toistuvan ja ajantasaisen datan keruun.

3.6.2 Vaihe 2: Yhteyden muodostaminen VMwareen

Käynnistetty Python-skripti muodostaa ohjelmallisen yhteyden VMware-ympäristön vSphere API-rajapintaan käyttäen pyvmomi-kirjastoa. Skripti autentikoi itsensä tarvittavilla tunnistetiedoilla ja hakee haluttujen virtuaalikoneiden tilatiedot, kuten suorittimen kuormitus, muistinkäyttö ja levytilan määrä.

3.6.3 Vaihe 3: Datan käsittely ja muotoilu

Datan käsittelyssä yksikköjä muutetaan yleisesti luettavampaan muotoon, tiedot ryhmitellään loogisesti ja lopputulos tallennetaan taulukkomuotoon, kuten CSV- tai Excel-tiedostoksi, joka soveltuu helppoon jatkoanalyysiin tai raportointiin.

3.6.4 Vaihe 4: Tiedoston autentikointi ja vienti Google Driveen

Kun data on käsitelty, skripti hyödyntää Google Drive API:a sekä google-auth-kirjastoa varmistaakseen turvallisen yhteyden käyttäjän Google Drive-tiliin. Tämän jälkeen valmis taulukko ladataan automaattisesti ennalta määritettyyn kansioon Google Drivessa. Tiedosto voidaan nimetä dynaamisesti, esimerkiksi päivämäärän perusteella, jolloin jokaisesta ajosta muodostuu oma versio.

3.6.5 Vaihe 5: Toiminnan toisto ja ylläpito

Crontabin avulla työnkulku käynnistyy uudelleen määritetyin aikavälein, mikä mahdollistaa jatkuvan ja automaattisen prosessin toistuvuuden ilman käyttäjän valvontaa. Kaikki vaiheet toimivat saumattomasti yhteen varmistaen, että ajantasainen ja eheä tieto on jatkuvasti saatavilla pilvipalvelusta.

4 PROJEKTIN TOTEUTUS

Tässä luvussa kuvaillaan tarkemmin projektin eri osien toimintaa. Luvussa käsitellään myös projektissa esiintyneitä haasteita, sekä ratkaisuja. Viimeisenä käsitellään projektin tuloksia. Skripti on nähtävissä kokonaisuudessaan liitteessä 1.

4.1 Projektin eri osien toteutus

Seuraavaksi käsitellään projektin eri osien toteutus osakokonaisuus kerrallaan. Yhdessä seuraavat osat muodostavat kokonaisen ja toimivan projektin.

4.1.1 Virtuaaliympäristön luominen

Projektin toteutuksessa hyödynnettiin Python-virtuaaliympäristöä, jonka avulla varmistettiin kehitysympäristön puhtaus ja riippuvuuksien hallinta. Virtuaaliympäristö mahdollistaa sen, että kaikki tarvittavat kirjastot ja paketit asennetaan erilliseen hakemistoon ilman, että palvelimen tai käyttäjän globaaleja Python-asennuksia tarvitsee muokata. Tämä parantaa niin projektin siirrettävyyttä kuin hallittavuutta ja helpottaa myöhempää ylläpitoa. Lisäksi tietoturvan parantamiseksi skripti käyttää dotenv-kirjastoa, jolla voidaan hyödyntää erillistä tunnuksia ja muuttujia sisältävää virtuaaliympäristöön sidottua ympäristötiedostoa. Dotenv-esimerkkitiedosto on nähtävissä liitteessä 2.

Virtuaaliympäristö luotiin seuraavien vaiheiden mukaisesti:
1. Virtuaaliympäristön alustaminen: Virtuaaliympäristö luotiin projektihakemistoon venv-moduulilla ajamalla komentorivillä:
python3 -m venv venv

2. Ympäristön aktivointi:
source venv/bin/activate

3. Riippuvuuksien asennus ja pakettien hallinta:
Kaikki projektissa tarvittavat Python-kirjastot määriteltiin requirements.txt-tiedostoon. Tämä mahdollistaa sekä ympäristön nopean uudelleenasennuksen että tarvittavien kirjastoversioiden hallinnan.

Testauksen jälkeen käytettyjen kirjastojen ja niiden versioiden lukitus tehtiin komennolla: pip freeze > requirements.txt
Kaikki riippuvuudet on mahdollista tämän jälkeen asentamaan yhdellä komennolla: pip install -r requirements.txt
Tämän avulla voidaan varmistaa, että ympäristö pysyy identtisenä myös ympäristöä siirrettäessä tai järjestelmää päivitettäessä.

Virtuaaliympäristön hyödyntäminen nopeutti käyttöönottoa ja varmisti, että kaikki projektin osat toimivat yhtenevästi riippumatta siitä, missä palvelimella tai käyttöympäristössä skriptiä ajettiin.

4.1.2 VMware-datan kerääminen

Virtuaalikoneympäristön tilatiedon keruu toteutettiin hyödyntämällä pyvmomi-kirjastoa, joka mahdollistaa ohjelmallisen integraation VMware vSphere -infrastruktuuriin. Skriptissä yhteyden muodostaminen tapahtui ympäristömuuttujista luettavilla tunnistetiedoilla, joka lisää turvallisuutta ja helpottaa ympäristöjen vaihtelua ilman koodimuutoksia.
Yhteys VMware-palvelimeen muodostetaan connect_to_vcenter-funktiolla, joka hyödyntää pyVmomi-paketin SmartConnect-metodia. Tietojen keruu rajautuu valittuihin vCenter-instansseihin, joihin yhdistetään vuorotellen.

```
def connect_to_vcenter(host, user, password, port):
    service_instance = SmartConnect(host=host, user=user, pwd=password, port=vm_port)
    return service_instance
```

Virtuaalikoneiden selaus ja datan kerääminen toteutetaan get_all_vms-funktiolla. Tämä funktio muodostaa vSphere-palvelimelle näkymän kaikista virtuaalikoneista, minkä jälkeen jokaisesta koneesta haetaan halutut tiedot.

```
vm_view = content.viewManager.CreateContainerView(content.rootFolder,
                                                  [vim.VirtualMachine],
                                                  True)
    obj = [vm for vm in vm_view.view]
    vm_view.Destroy()
    print("Virtual machines found:")
```

```
    print(obj)
    return obj
```

Myös virtuaalikoneisiin liittyvät yksityiskohtaiset tiedot, kuten isäntäkoneen IP-osoite (get_host_ip), DNS-nimi (get_dns_name) ja verkkoliitäntöihin liittyvät tiedot (get_network_info) kerätään erillisillä apufunktioilla. Tämä mahdollistaa tuloksen laajentamisen ja muokkaamisen myös jatkokehitystä ajatellen.
Kerätty raakadata kulkee selkeästi vaiheittain, jolloin jokainen tiedonkeruufunktio voidaan tarvittaessa testata ja käyttää uudelleen ilman koko skriptin suoritusta. Lopuksi yhteys VMware-palvelimeen suljetaan Disconnect-komennolla virheiden ja resurssien tuhlauksen välttämiseksi.

4.1.3 Datan käsittely

Kun VMware-ympäristöstä kerätyt tiedot on poimittu, suoritetaan niiden jatkokäsittely, jonka tavoitteena on tuottaa selkeä ja raportointiin soveltuva taulukko. Datan muotoilussa jokaisen virtuaalikoneen tiedot asetetaan omalle rivilleen. Sarakkeisiin sijoitetaan esimerkiksi koneen nimi, isäntäkoneen IP-osoite, resurssipooli, verkko-osoitteet, prosessorien määrä, muistin määrä ja muut metatiedot. Tarvittaessa puuttuvat arvot täydennetään tai käsitellään, ja tietotyypit muunnetaan yhdenmukaisiksi, jotta tiedostosta tulee helposti luettava ja analysoitava.

Datan käsittely ja tallennus Google Sheetsiin tapahtuu käytännössä luomalla taulukkotiedosto, johon lisätään haetut tiedot. Taulukon muotoilu on tehtävä koodin puolella käyttäen taulukon filtteröinti- tai request-komentoja.
Lopputuloksena tuotettu taulukko siirretään Google Sheets -asiakirjaan ja jaetaan haluttuun kansioon Google Driveen jatkokäyttöä varten.

4.1.4 Google Drive-integraatio

Datan käsittelyn jälkeen projektin tärkeä vaihe on tietojen siirtäminen Google Driveen, jossa ne ovat helposti saavutettavissa organisaation eri käyttäjille ja sidosryhmille. Tiedonsiirto toteutetaan Google Drive API:n sekä Google Sheets API:n avulla automatisoidusti suoraan Python-skriptistä. Tämä eliminoi manuaalisten tallennus- ja latausvaiheiden tarpeen ja varmistaa, että data on aina ajantasaisesti tallessa pilvessä.

Ennen tiedostojen vientiä Google Driveen skripti huolehtii käyttäjän tunnistuksesta ja valtuutuksesta OAuth 2.0 -protokollan mukaisesti. Kun oikeudet on myönnetty, skripti voi sekä luoda että päivittää taulukoita ja muita tiedostoja määriteltyyn Drive-kansioon.

Tiedonsiirron yhteydessä skripti:
Luo uuden taulukon tai päivittää olemassa olevan Google Sheets -asiakirjan määriteltyyn kansioon. Siirtää kaikki käsitellyt tiedot automaattisesti riveittäin Google Sheetin taulukkoon. Varmistaa, että tiedoston jakaminen ja käyttöoikeudet ovat Drive-ympäristössä organisaation tarpeiden mukaiset.
Integraation ansiosta tiedot ovat nopeasti saatavilla, helposti hyödynnettävissä ja niiden versiohistoria on mahdollista säilyttää Google Driven ominaisuuksien avulla. Automatisoitu tiedonsiirto parantaa tiedon hallintaa, varmistaa ajantasaisuuden ja helpottaa raportointia ilman erillisiä eräajoja tai käsin suoritettavia toimia.

4.1.5 Automaatio Crontabin avulla

Järjestelmän toiminta perustuu täysautomaattiseen työnkulkuun, jossa Python-skripti suoritetaan säännöllisesti ilman käyttäjän manuaalista puuttumista. Tämän mahdollistaa Crontab, Linux-ympäristön ajastus- ja tehtävienhallintatyökalu. Crontab:iin määritetty ajastus varmistaa, että tiedonkeruu, käsittely ja vienti Google Driveen tapahtuvat määräajoin esimerkiksi tunneittain, päivittäin tai viikoittain tarpeiden mukaisesti.
Automaatio Crontabin avulla toteutettiin seuraavasti:
Määriteltiin ajastusmerkintä Crontab-tiedostoon komentorivillä esimerkiksi komennolla crontab -e
Lisättiin ajoitusrivi, joka käynnistää Python-skriptin haluttuna ajankohtana. Yleinen muoto on: 0 3 * * * /polku/skriptiin/venv/bin/python /polku/skriptiin/main.py
Tämä esimerkki suorittaa skriptin päivittäin klo 3.00.
Ajastus testattiin simuloimalla crontabin toimintaa ja seuraamalla järjestelmän lokitietoja sekä Google Driveen päivittyviä tiedostoja.

4.2 Haasteet ja ratkaisut

Seuraavaksi käsitellään projektin aikana syntyneitä haasteita sekä löydettyjä ratkaisuja. Projektissa on useita eri järjestelmiä, joiden versiot ja käytetyt rajapintaratkaisut vaikuttavat toisiinsa.

4.2.1 Teknologiset haasteet

Projektin suurimmat teknologiset haasteet liittyivät oikeiden ja yhteensopivien ohjelmistokomponenttien löytämiseen sekä eri rajapintojen toimivuuden varmistamiseen. Erityisesti sekä VMware- että Google Drive -integraatiot vaatineet huolellista suunnittelua ja ratkaisujen testausta, jotta tietojen keruu ja siirto saatiin toteutettua tehokkaasti ja luotettavasti.

Yksi keskeinen haaste liittyi VMware vSphere API:n ja pyvmomi-kirjaston yhteensopivuuteen. Eri VMware-ympäristöillä ja vSphere-versioilla oli omat erityispiirteensä, minkä vuoksi tuli varmistaa, että käytetty pyvmomi-versio tukee kyseistä ympäristöä ja API:n komentoja. Dokumentaatio oli osittain hajanaista, ja oikean komentorakenteen löytämiseen kului aikaa.

Vastaava yhteensopivuushaaste kohdistui Google Drive API:n ja Google Sheets API:n versioihin ja kirjastoihin. Google API -kirjastot päivittyvät säännöllisesti, minkä seurauksena osa komentorakenteista, autentikoinnin tavoista ja tiedonsiirron muodoista muuttuu. Oli tärkeää seurata sekä Google API-kirjastojen että käyttöoikeuksien (OAuth 2.0-virtaus) ajantasaisuuksia, jotta tiedonsiirto Google Driveen sujui moitteettomasti.

Lisäksi tehokkaan tiedonkeruun ja tiedonsiirron toteutus vaati skriptin optimointia ja tiedon käsittelyn suunnittelua. Oli tärkeää varmistaa, että tietojen haku VMware-ympäristöstä ja niiden vienti Google Driveen tapahtuivat loogisessa järjestyksessä ilman turhia odotuksia, virheitä tai tiedon katoamista.
Yhteenvetona voidaan todeta, että haasteet liittyivät pääosin versionhallintaan sekä rajapintojen dokumentaatioon. Näiden ratkominen edellytti sekä teknistä syventymistä eri teknologioiden dokumentaatioon että testaamista käytännön kokeiluilla.

4.2.2 Toteutetut ratkaisut

Teknologisiin haasteisiin vastaamiseksi projektissa hyödynnettiin systemaattista testaamista ja perusteellista dokumentaatioihin perehtymistä. Ensimmäinen askel oli kartoittaa käytössä olevien VMware- ja Google API-versioiden sekä Python-kirjastojen yhteensopivuudet. Koska sekä pyvmomi- että Google API -kirjastot kehittyvät nopeasti ja niiden rajapintatoiminnot voivat vaihdella eri versioissa, yhteensopivuus varmistettiin asentamalla ja testaamalla kirjastojen eri versioita paikallisesti virtuaaliympäristössä.

Manuaalista yhteensopivuuden testausta tehtiin vaiheittain: yksittäiset komennot ja funktiokutsut testattiin pienissä testiskripteissä ennen niiden ottamista osaksi lopullista kokonaisskriptiä. Tämä auttoi tunnistamaan nopeasti mahdolliset virheilmoitukset ja selvittämään, mitkä komentorakenteet tai autentikointimenetelmät toimivat kullakin ohjelmistoversiolla.

Lisäksi projektissa luettiin huolellisesti sekä VMware-ympäristön että Google Driven ohjelmisto- ja rajapintadokumentaatiota. Erityisesti authentication- eli valtuutusprosessin sekä tiedostojen siirtoon liittyvien funktioiden käytössä dokumentaatio oli ratkaiseva apu toimivien esimerkkien ja best practices-menetelmien löytämiseksi. Käytännön kokeiluja täydennettiin tarvittaessa myös yhteisön foorumien, Github-keskustelujen ja Stack Overflow'n ohjeistuksilla.

Yhteenvetona haasteet ratkaistiin iteratiivisella lähestymistavalla, jossa ensin tunnistettiin rajapintojen toiminnallisuudet, tämän jälkeen kokeiltiin ja testattiin vaihtoehtoisia komentorakenteita, ja lopulta dokumentaatiota ja testituloksia hyödyntäen varmistettiin toimiva kokonaisuus. Tämä varmistaa pitkällä aikavälillä järjestelmän ylläpidettävyyden ja mahdollistaa joustavan päivittämisen myös rajapintojen uudistuessa.

4.3 Tulokset

Tuloksissa kuvaillaan projektin eri osien tuottamia tuloksia. Tuloksia tarkastellaan Järjestelmän suorituskyvyn, datan validoinnin, käyttäjähyötyjen sekä esimerkin avulla.

4.3.1 Järjestelmän suorituskyky

Järjestelmän suorituskyky arvioitiin tarkastelemalla Python-skriptin ajonaikaista tehokkuutta, resurssien käyttöä sekä vasteaikoja eri ympäristöissä. Suunnitellun ratkaisun etuna on, että skripti ei vaadi merkittävää laskentatehoa tai muita laitteistoresursseja – kaikki toiminnallisuus perustuu kevyiden ohjelmointikirjastojen ja API-kutsujen käyttöön.

Varsinaisessa käytössä skriptin suoritus on hyvin nopeaa: koko työnkulku, joka sisältää tiedonkeruun VMware-ympäristöstä, datan käsittelyn sekä tiedostojen siirron Google Driveen, vie tyypillisesti vain muutamia sekunteja. Suorituskyky riippuu ensisijaisesti verkon nopeudesta ja yhteyksien kuormituksesta – esimerkiksi hyvin vilkkaaseen aikaan tai hitailla etäyhteyksillä ajonaika voi jonkin verran pitkittyä. Kuitenkin suoritus on pysynyt jatkuvasti odotetulla tasolla, eikä raskaampaa kuormitusta tai resurssipulaa ole havaittu.

Skriptin kevyt rakenne mahdollistaa sen ajamisen niin palvelinympäristöissä. Käytetyllä Linux-alustalla on useita muitakin ajastettuja töitä, joiden samanaikainen ajaminen voisi hetkellisesti lisätä palvelimen kuormitusta. Tämän vuoksi resurssien käytön optimointi huomioitiin porrastamalla tämän skriptin ajastus siten, ettei suoritus ajoitu samalle ajankohdalle muiden suuria resursseja vaativien prosessien kanssa. Näin varmistettiin sekä järjestelmän tasainen suorituskyky että muiden palveluiden häiriötön toiminta. Kokonaisuutena järjestelmä täyttää vaaditut suorituskyvyn kriteerit, ja ratkaisu on osoittanut luotettavuutensa ja tehokkuutensa puolen vuoden käytössäoloaikana.

4.3.2 Datan validointi

Järjestelmän tuottaman datan tarkkuus ja paikkansapitävyys ovat olennaisia edellytyksiä automaattisen tiedonkeruuprosessin luotettavuudelle. Tässä projektissa datan tarkkuus validoitiin vertaamalla skriptin tuottamia taulukoita suoraan VMware-hallintaympäristön tarjoamiin tietoihin.

Validointiprosessin aikana satunnaisesti valittiin useita virtuaalikoneita ja niiden tiedot tarkistettiin yksitellen sekä VMware-hallintapaneelista että Google Sheet-taulukosta, johon järjestelmä tiedot automaattisesti kerää ja tallettaa.

Kaikkien tarkasteltujen tietojen osalta automaattisesti muodostettu taulukko vastasi VMware-hallinnasta löytyneitä tietoja. Lisäksi tiedot olivat siirtyneet yhdenmukaisesti ja oikeassa muodossa Google Driveen asetettuun taulukkoon ilman olennaisia poikkeamia tai virheitä. Yksittäisten virtuaalikoneiden tiedot näkyivät täsmällisesti myös mahdollisten muutosten jälkeen, mikä osoittaa järjestelmän reaaliaikaisen ja ajantasaisen päivittymiskyvyn.
Datan validointi satunnaisotannalla toi lisää varmuutta järjestelmän toimivuuteen ja takasi, että automatisoidun työnkulun tuottama data on käyttökelpoista.

4.3.3 Käyttäjähyödyt

Kehitetyn järjestelmän keskeiset käyttäjähyödyt liittyvät erityisesti tiedon saatavuuteen ja ongelmanselvityksen tehokkuuteen. Kun virtuaalikoneiden tiedot on koottu automaattisesti yhteen, ajantasaiseen taulukkoon, voidaan tarvittava tieto löytää nopeasti ilman tarvetta manuaaliselle selaamiselle VMware-hallintajärjestelmässä.

Erityisesti ongelmanselvitystilanteissa koottu taulukkomuoto osoittautui hyödylliseksi. Tyypillisesti palveluiden ylläpidossa tarvitaan nopeasti tarkasteltavaksi esimerkiksi yksittäisen virtuaalikoneen host-palvelin, käytössä olevat resurssit (kuten prosessorit, muisti ja levytila) sekä mahdolliset verkkoliitännät. Kun kaikki olennainen tieto on saatavilla selkeästi jäsenneltynä ja keskitetysti Google Driven taulukossa, manuaalisen työn ja tiedon etsimisen tarve vähenee merkittävästi.

Ratkaisu vapauttaa erityisesti ylläpitohenkilöstön aikaa, parantaa järjestelmän läpinäkyvyyttä ja mahdollistaa ripeämmän päätöksenteon sekä reagoinnin mahdollisiin poikkeamatilanteisiin. Lisäksi tiedon keskitetty saavutettavuus tukee tiimityötä ja raportointia, kun useat työntekijät pääsevät ajantasaisen datan äärelle samanaikaisesti.

Järjestelmässä on myös toteutettu automaattinen tiedostojen versiointi. Python-skriptin logiikka varmistaa, että Google Drivessa säilytetään vain kolme viimeisintä versiota taulukosta. Vanhemmat versiot poistetaan automaattisesti uuden version luonnin yhteydessä. Tämä toimintatapa auttaa pitämään tallennustilan

hallittuna sekä ehkäisee vanhentuneen datan kertymistä. Samalla se mahdollistaa paluun aikaisempiin datoihin lyhyellä aikajänteellä esimerkiksi virhetapausten tai muutospyyntöjen yhteydessä.

Ratkaisun ansiosta järjestelmän käyttäjillä on aina saatavillaan tuorein tieto sekä mahdollisuus palauttaa tuoreimpia aiempia taulukoita tarvittaessa.

4.3.4 Esimerkki

Alla olevaan taulukkoon on koottu esimerkki järjestelmän automaattisesti keräämästä virtuaalikoneiden tiedostosta. Taulukko havainnollistaa, miten skripti kokoaa kunkin virtuaalikoneen oleelliset tiedot kuten hostin, resurssien käytön, verkko-osoitteet ja muut keskeiset attribuutit yhteen näkymään Google Driveen. Taulukkoon on poimittu kolmesta esimerkkikoneesta kerätyt ja tallennetut tiedot. Lisäksi esimerkkinä VM 3, joka on pois käytöstä. Taulukon data anonyymiä esimerkkidataa tietoturvasyistä.

TAULUKKO 1. Anonymisoitu esimerkkitaulukko tuloksista

Attribute	**VM 1**	**VM 2**	**VM 3**
Host IP	internal.host.ip.address	external.host.ip.address	internal.host.ip.address
VM Name	Network Boot Server	Public Gitlab	Alarm Test
DNS Name	networkboot.network.com	gitlab.network.com	VMware tools not running
Resource Pool	Maintenance	Public Infra	Test
Network Adapters and IP Addresses	[{"network": "Company-Network", "ip_addresses": ["ip.address.to.vm"]}	[{"network": "Public-Network", "ip_addresses": ["ip.address.to.vm"]}	[{"network": "Company-Network", "ip_addresses": [""]}
CPU Count	1	2	1
Memory (MB)	2048	8192	2048
Total Hard Disk Size (GB)	40	200	40
Power State	PoweredOn	PoweredOn	PoweredOff
VM Note	Server hosting network boot installation tools and media	Server hosting public gitlab instance	Alarm system testing (maintenance dep.)

5 TESTAUS JA VALIDOINTI

Tässä luvussa käsitellään projektissa käytettyjä testausmenetelmiä, testauksen tuloksia, projektiin sisäänrakennettua virheidenkäsittelyä sekä käyttäjäpalautetta. Testiskenaarioissa ja tuloksissa on kuvailtuna erilaisia virhetilanteita, käyttötapauksia sekä vastaavien tilanteiden tulokset. Virheidenkäsittelyssä kuvataan tapoja, joilla skriptiin on rakennettu tapoja käsitellä ja toipua virhetilanteista. Lopuksi käyttäjäpalautteessa on työssä tehtyjä havaintoja projektin vaikutuksista.

5.1 Testiskenaariot ja tulokset

Järjestelmän toimivuuden ja luotettavuuden varmistamiseksi suoritettiin useita eri testiskenaarioita, jotka kattavat sekä tiedonkeruun, -käsittelyn että -siirron Google Driveen. Testauksen tavoitteena oli tunnistaa mahdolliset virhe- ja poikkeustilanteet sekä varmistaa datan eheyden ja saavutettavuuden kaikissa tilanteissa.
Testauksen perusteella järjestelmän suorituskyky, tietojen keruu ja tiedonsiirto Google Driveen toimivat suunnitellulla tavalla useimmissa testatuissa skenaarioissa. Skripti onnistui hakemaan ja käsittelemään tiedot erilaisten virtuaalikoneiden kokoonpanoista sekä siirtämään ne luotettavasti Google Driveen ajastusten mukaisesti.

Testiskenaariot suunniteltiin kattamaan muun muassa seuraavat osa-alueet:

5.1.1 Datan keruu erilaisista VM-konfiguraatioista:

Testattiin skriptin kyky hakea ja käsitellä tietoja erilaisista virtuaalikoneista, joilla oli vaihtelevia resursseja (prosessorit, muisti, levytila) ja erilaisia verkkoratkaisuja. Näin varmistettiin, että skripti toimii sekä yksinkertaisten että monimutkaisten VM:n kanssa.
Merkittävä havainto testauksesta oli, että skripti suoriutui hyvin myös tilanteissa, joissa yksittäisestä virtuaalikoneesta puuttui tietoja, kuten osa verkkoliitännöistä tai metatiedoista. Skriptiin toteutetun virheenkäsittelyn ansiosta puuttuvat tai virheelliset tiedot eivät estäneet tiedonkeruun onnistumista. Sen sijaan järjestelmä tulosti käyttäjälle selkeän virheilmoituksen puuttuvasta tiedosta tai poikkeavasta tilanteesta, minkä ansiosta käyttäjä pystyy reagoimaan poikkeamiin tarvittaessa nopeasti.

5.1.2 Epäonnistuneet API-kutsut:

Tarkoituksella simuloitiin tilanteita, joissa VMware vSphere API:in tai Google Drive API:in ei saatu yhteyttä (esim. virheelliset kirjautumistiedot, palvelimen tilapäinen katkos). Testattiin, pysyykö skripti hallitussa tilassa virheen sattuessa sekä tuottaako se informatiivisen virheilmoituksen ilman tietojen katoamista.
Epäonnistuneiden API-yhteyksien ja verkkoyhteysongelmien tapauksessa skripti pysyi hallitussa tilassa ja tuotti ymmärrettäviä virheilmoituksia ilman, että koko työnkulku päätyi selittämättömästi. Ajastetut suoritukset crontabin kautta toteutuivat säännöllisesti, eikä päällekkäisiä ajoja tai järjestelmähäiriöitä ilmennyt.
Kaiken kaikkiaan testitulokset osoittivat, että järjestelmä on vakaa, kestää poikkeamatilanteita ja kykenee käsittelemään erilaisia virhetilanteita ilman olennaista tietojen menetystä tai prosessin keskeytymistä.

5.1.3 Crontabin ajoitettujen tehtävien ajo:

Testattiin, käynnistyvätkö skriptit crontabin ajamina halutun aikataulun mukaisesti ja siirtyvätkö tiedostot Google Driveen jokaisen ajon jälkeen. Lisäksi seurattiin, että ajoitus ei aiheuttanut häiriöitä muihin samanaikaisiin palvelintehtäviin.
Näiden testiskenaarioiden avulla pyrittiin varmistamaan, että järjestelmä suoriutuu suunnitelluista tehtävistään virheettömästi eri käyttö- ja poikkeustilanteissa, ja että data säilyy eheänä myös mahdollisten virheiden ilmaantuessa.

5.2 Virheidenkäsittely

Luotettavan automaatiojärjestelmän kannalta virheiden käsittelyllä on keskeinen merkitys, jotta yksittäiset poikkeamat tai odottamattomat tilanteet eivät katkaise koko työnkulkua. Tässä projektissa skripti toteutettiin siten, että se tunnistaa vakiintuneet virhetilanteet (esimerkiksi puuttuvat yhteydet, API-kutsujen epäonnistumiset tai tiedon puuttumiset) ja osaa käsitellä ne järkevästi.

5.2.1 Tiedon puuttuminen ja virheiden ohittaminen:

Koodia suunniteltaessa huomioitiin sellaiset tilanteet, joissa virtuaalikoneelta puuttuu esimerkiksi verkko-osoitteita, isäntätietoja tai muita metatietoja. Skriptissä käytettiin try-except-rakenteita poikkeuksien varalta – esimerkiksi get_dns_name(vm)-funktiossa määriteltiin, että jos DNS-nimeä ei saada tai VMware Tools ei ole käytössä, palautetaan informatiivinen oletusarvo (“VMware Tools not running”, “DNS name not available” tai “Error”).

5.2.2 API-yhteyksien ja verkkovirheiden käsittely:

Jos yhteyden muodostaminen VMware-ympäristöön (SmartConnect) ja Google Driveen (OAuth 2.0) epäonnistuu, skripti pystyy tulostamaan käyttäjälle eritellyn virheilmoituksen esimerkiksi palvelimen saatavuudesta, vääristä tunnuksista tai aikakatkaisusta. Tämä helpottaa ylläpitäjää tunnistamaan ongelmat nopeasti ja estää tilanteen, jossa koko prosessi pysähtyisi odottamatta.

5.2.3 Virheilmoitusten tulostus käyttäjälle:

Skriptissä on johdonmukaisesti huomioitu virheilmoitusten tulostaminen komentoriville (esim. print(f"Error retrieving DNS name for VM: {str(e)}")). Näin käyttäjä saa ajantasaisen tiedon siitä, missä vaiheessa ja minkä VM:n tietojen haussa mahdollinen virhe on tapahtunut. Vastaavaa logiikkaa sovelletaan myös tiedostojen käsittelyssä ja Google Driven API-toiminnoissa.

5.2.4 Yleinen vakautus ja jatkokäsittely:

Kokonaisuudessaan ratkaisu mahdollistaa sen, että yksittäiset virhetilanteet eivät vaikuta muiden virtuaalikoneiden tiedonkeruuseen. Prosessi jatkuu niiltä osin kuin mahdollista ja puuttuvat tiedot osoitetaan selkeästi lopullisessa tulostaulukossa. Näin järjestelmä pysyy toimintakykyisenä ja ongelmat voidaan jälkikäteen tunnistaa sekä korjata joko käsin tai kehittämällä jatkossa uusia tarkistuksia skriptiin.

5.3 Käyttäjäpalautteet

Järjestelmän käyttöönoton jälkeen palautetta on saatu ylläpidosta vastaavalta henkilöstöltä, joka hyödyntää automaattisesti tuotettua taulukkoa työssään. Käyttäjäpalautteen perusteella ratkaisu on otettu vastaan erittäin myönteisesti. Ylläpidon kannalta suurimmaksi hyödyksi on osoittautunut se, että ajantasainen ja keskitetysti koottu taulukko on jatkuvasti saatavilla ilman manuaalisia tiedonkeruuprosesseja.

Taulukko on ollut erityisen hyödyllinen silloin, kun on selvitetty palvelinympäristön ongelmatilanteita, tutkittu yksittäisen virtuaalikoneen resursseja tai etsitty nopealla aikataululla tiettyyn koneeseen liittyviä tietoja. Ennakkoon järjestetyn ja auto-

maattisesti päivittyvän tietokannan ansiosta ylläpito on kyennyt reagoimaan poikkeamiin nopeammin, vähentämään virheiden määrää ja löytämään olennaisen tiedon ilman turhia välivaiheita.

Yhteenvetona voidaan todeta, että käyttäjät kokevat ratkaisun parantaneen työn sujuvuutta, tiedon löydettävyyttä sekä valvonta- ja selvitystyön tehokkuutta. Automatisoitu ratkaisu koetaan luotettavana ja helposti omaksuttavana osana ylläpitäjien arkea.

6 YLLÄPITO JA TULEVAISUUDEN PARANNUKSET

Tässä luvussa käsitellään projektin ylläpitovaatimuksia sekä mahdollisia tulevaisuuden parannuksia. Opinnäyteyön kirjoitushetkellä järjestelmä toimii omana eristettynä tehtävänään, joten se ei vaadi jatkuvaa ylläpitoa. Järjestelmään vaikuttavat tällä hetkellä ainoastaan rajapintojen käyttämien järjestelmien muutokset. Lisäksi tarkastellaan järjestelmän etuja sekä mahdollisia parannuksia tulevaisuuteen.

6.1 Pitkän aikavälin ylläpito

Järjestelmän pitkäaikainen toimivuus ja luotettavuus edellyttävät säännöllistä ylläpitoa sekä projektin eri komponenttien ajantasaisuuden seurantaa. Koska ratkaisu rakentuu useista eri teknologioista ja rajapinnoista – kuten VMware vSphere API, pyvmomi, Google OAuth sekä Google Drive API – on tärkeää olla tietoinen jokaisen komponentin päivityksistä ja mahdollisista käyttötapojen muutoksista.

Ylläpidon keskeisiä tehtäviä ovat muun muassa:

6.1.1 Ohjelmistopäivitysten ja muutosten seuraaminen

Jos jokin projektin käyttämä ulkoinen palvelu, kuten VMware-järjestelmä tai Googlen OAuth-valtuutus, päivittyy tai muuttaa rajapintaansa, on järjestelmän toiminta testattava päivitysten jälkeen. Tarvittaessa myös skriptin riippuvuuksia ja komentorakenteita tulee päivittää yhteensopivuuden säilyttämiseksi.

6.1.2 Python-riippuvuuksien hallinta

Käytettävien Python-kirjastojen (esim. pyvmomi, google-auth, google-api-python-client) versiot tulee tarkistaa säännöllisesti ja päivittää hallitusti tarpeen mukaan. Päivitysten jälkeen on hyvä suorittaa kattavat testit sen varmistamiseksi, että järjestelmä toimii kuten ennenkin.

6.1.3 API-avainten ja autentikointitietojen ylläpito

Google Drivelle, VMwarelle ja muille mahdollisille integraatioille käytettävät avaimet ja tunnistetiedot vanhenevat tyypillisesti määräajoin. Ylläpidon tulee huolehtia niiden uusimisesta ja päivittämisestä sekä hallita käyttöoikeuksia asianmukaisesti.

6.1.4 Toiminnan säännöllinen testaus

Erityisesti isojen päivitysten tai järjestelmien muutosten yhteydessä on tärkeää suorittaa kokonaisuuden testaus käytännön ajossa. Näin mahdolliset yhteensopivuusongelmat voidaan havaita ja korjata ennen tuotantokäytön häiriintymistä.

6.1.5 Dokumentoinnin päivitys

Jokainen komponenttien päivitys tai järjestelmän toimintatapojen muutos tulisi dokumentoida selkeästi. Hyvin ylläpidetty dokumentaatio helpottaa tulevien ylläpitäjien ja kehittäjien työtä.

Tällä hetkellä projekti toimii omassa rajatussa virtuaaliympäristössään. Näin ollen järjestelmä ei vaadi aktiivisia ylläpitotoimenpiteitä ennen kuin ulkopuoliset järjestelmät – kuten VMware-alusta, Google Drive -ympäristö tai näihin liittyvät rajapinnat – päivittyvät tai niiden käytössä tapahtuu olennaisia muutoksia. Käytännössä ylläpidon tarve korostuu erityisesti silloin, kun jokin ulkoinen komponentti muuttuu, mutta vakaassa omassa ympäristössään järjestelmä toimii ilman jatkuvaa valvontaa.

6.2 Ylläpitotoiminnan ohjeistus ja dokumentointi

Järjestelmän tehokkaan ylläpidon sekä jatkuvan toiminnan varmistamiseksi on tärkeää, että keskeiset ylläpitotoimenpiteet, kokoonpanot, päivitysprosessit ja mahdolliset ongelmatilanteiden ratkaisumallit on dokumentoitu selkeästi. Projektin ylläpidon ohjeistus on järjestetty hyödyntämällä organisaation käytössä olevia sisäisiä dokumentointialustoja, joihin on tallennettu vaiheittaiset toimintaohjeet ja vastuuhenkilöt.

Näillä dokumentaatioalustoilla ylläpidetään muun muassa:

Virtuaaliympäristön rakenteet (esimerkiksi palvelinosoitteet, käyttäjätunnisteet, käytetyt rajapinnat ja niihin liittyvät käyttöoikeudet), jotka on kirjattu tarkasti ja ovat saatavilla vain määritellyille vastuuhenkilöille tietoturvan varmistamiseksi.

Skriptin toimintalogiikka ja ajastus (crontab-merkinnät, ympäristömuuttujien sijainnit, tarvittavat päivitysprosessit) on dokumentoitu vaihe vaiheelta helposti seurattavassa muodossa.

Tyypilliset virhetilanteet ja niiden ratkaisut on koottu esimerkiksi ohjeistuksiin tai taulukoihin, joiden avulla ylläpitäjät voivat nopeasti tunnistaa ongelman ja toimia sen mukaisesti.

Lokitiedostojen sijainnit ja järjestelmän oma seuranta on kuvattu niin, että ylläpitäjällä on selkeä käsitys järjestelmän tilasta ja reagointimahdollisuudet tapahtumiin.

Ajantasainen dokumentaatio on avainasemassa järjestelmän elinkaaren jokaisessa vaiheessa. Hyvin ylläpidetyt ohjeet tukevat uuden ylläpitäjän perehdytystä, helpottavat vikojen selvittämistä sekä mahdollistavat ylläpitovastuun siirtämisen organisaatiossa ilman toimintakatkoksia. Lisäksi selkeä ja saavutettava dokumentaatio on edellytys mahdollisille järjestelmän laajennuksille, integraatioille tai jatkokehitykselle tulevaisuudessa.

6.3 Seuranta ja hälytykset

Järjestelmän luotettavuuden ja jatkuvan toiminnan varmistamiseksi on tärkeää, että mahdolliset virhetilanteet ja poikkeamat havaitaan nopeasti. Tässä projektissa seuranta on toteutettu sekä skriptiin rakennetun virheilmoitusten tulostuksen avulla että hyödyntämällä Linux-palvelinta valvovia seuranta- ja hälytysjärjestelmiä.

Skripti on suunniteltu tuottamaan selkeitä virheilmoituksia komentoriville silloin, kun tiedonkeruussa, datan käsittelyssä tai tiedonsiirrossa ilmenee poikkeamia. Ylläpidon näkökulmasta tämä mahdollistaa reagoinnin mahdollisiin ongelmiin heti, kun ajon aikana havaitaan puutteita tiedoissa, virheellisiä yhteyksiä tai muita teknisiä häiriöitä. Virheilmoitukset dokumentoivat myös, missä vaiheessa ja minkä virtuaalikoneen kohdalla ongelma on ilmennyt, mikä helpottaa jälkiselvitystä.

Lisäksi Linux-palvelin, jolla automaatioskripti ajetaan, tarjoaa käyttöön monipuoliset työkalut järjestelmän tilan seurantaan ja tarvittaessa hälytysten lähettämiseen. Tällaisia työkaluja ovat esimerkiksi syslog, cronin sähköposti-ilmoitukset, palvelimen resurssimonitorit sekä erilliset käytössä olevat valvontasovellukset.

Näiden avulla on mahdollista saada automaattinen ilmoitus esimerkiksi epäonnistuneesta suorituksesta, levytilan loppumisesta, verkko-ongelmista tai muista kriittisistä poikkeamista.

Kokonaisuutena yhdistetty skriptipohjainen virheidenhallinta ja palvelintason hälytyslogiikka varmistavat, että järjestelmän ylläpito pysyy ajan tasalla mahdollisista ongelmista ja voi reagoida niihin nopeasti.

6.4 Skaalautuvuus

Toteutetun järjestelmän yksi merkittävimmistä teknisistä eduista on sen hyvä skaalautuvuus. Skriptin ja työnkulun toiminta ei ole sidoksissa virtuaalikoneiden määrään, vaan kaikki virtuaaliympäristön tiedot haetaan ja käsitellään jokaisella ajokerralla. Näin ollen järjestelmä suoriutuu yhtä hyvin tilanteessa, jossa ympäristössä on vain muutama virtuaalikone, kuin silloin, kun virtuaalikoneiden määrä kasvaa, ainoastaan haettavaan tietomassaan käytettävä aika kasvaa maltillisesti. Skaalautuvuuden etuna on myös se, että mikäli VMware-ympäristöön luodaan uusia virtuaalikoneita, ne sisältyvät automaattisesti seuraavan ajoitetun suorituksen aikana tuotettuun taulukkoon. Käyttäjien ei siis tarvitse tehdä järjestelmässä erillisiä päivityksiä, vaan uusi tieto päivittyy Google Driven taulukkoon aina automaattisesti, kun uudet virtuaalikoneet on lisätty VMwareen ja automaatioskripti suoritetaan seuraavan kerran.

6.5 Mahdolliset lisäominaisuudet

Toteutettu ratkaisu tarjoaa perustan, jota voidaan laajentaa useilla hyödyllisillä lisäominaisuuksilla organisaation muuttuvien tarpeiden mukaan. Yksi potentiaalinen laajennusmahdollisuus on tiedon tallentaminen vaihtoehtoisiin säilytyspaikkoihin, kuten organisaation verkkotallennuslevylle tai muulle tiedostopalvelimelle Google Driven lisäksi. Skriptiä muokkaamalla on mahdollista esimerkiksi tallentaa tuotettu taulukko (CSV- tai Excel-muodossa) suoraan määritettyyn verkkojakoon automaattisen työnkulun yhteydessä. Tämä parantaa tiedon saavutettavuutta myös sellaisissa tilanteissa, joissa pilvipalveluiden käyttö on rajoitettua tai tarvitaan paikallisia varmuuskopioita.

Lisäksi Google Sheets -dokumentin ominaisuuksia voidaan hyödyntää entistä enemmän. Tiedot voidaan esimerkiksi rikastaa Dropdown-valikoilla, datan validoinneilla tai suodatustoiminnoilla Google Sheets -rajapinnan kautta. Näiden avulla käyttäjät voivat jatkossa helposti valita tai rajata näkymiä. Näiden ominaisuuksien hyödyntäminen nostaa käyttöliittymän interaktiivisuutta ja helpottaa tiedonhallintaa erityisesti laajoissa ympäristöissä.

Edellä mainitut laajennusmahdollisuudet voidaan toteuttaa joko lisäämällä tarvittavat Python-kirjastot ja -toiminnallisuudet nykyiseen skriptiin tai hyödyntämällä Google Sheetsin omia jatkokehitystyökaluja. Näin voidaan kehittää järjestelmästä entistä monipuolisempi ja mukautuvampi työväline organisaation tarpeisiin.

7 TODELLISET SOVELLUSKOHTEET

Tässä luvussa käsitellään projektin todellisia sovelluskohteita. Käyttötapauksissa on kuvailtu käytännön tilanteita, joissa projektin tuottama taulukointi helpottaa työskentelyä. Lisäksi tarkastellaan IT-alaa ja sen trendejä, joissa automaation merkitys on kasvavassa merkityksessä.

7.1 Käyttötapaukset

Käytännön esimerkkinä projektin hyödyllisyydestä on tilanne, jossa järjestelmän valvonta havaitsee virhetilanteen jossakin virtuaalikoneessa. Aiemmin ylläpitäjien oli tällaisessa tilanteessa selvitettävä virtuaalikoneen sijainti ja host-palvelin manuaalisesti, mikä saattoi olla aikaa vievää ja vaati siirtymistä useiden eri järjestelmien ja käyttöliittymien välillä. Automaattisesti luodun taulukon ansiosta tämä selvitys on nyt huomattavasti nopeampaa ja tehokkaampaa: ylläpitäjä voi tarkistaa suoraan ajantasaisesta taulukosta kyseisen virtuaalikoneen host-osoitteen ja kirjautua nopeasti oikeaan VMware-hallintajärjestelmään, mikä nopeuttaa virhetilanteen diagnosointia ja korjaamista olennaisesti.

Toisena esimerkkitapauksena on tilanne, jossa suunnitellaan uuden toiminnallisuuden tai ohjelmiston asentamista olemassa olevalle virtuaalikoneelle. Päätöksenteon tueksi tarvitaan ajantasainen tieto käytössä olevista resursseista, kuten muistista, prosessoritehosta ja levytilasta. Keskitetystä taulukosta nämä resurssitiedot löytyvät selkeästi jokaisesta virtuaalikoneesta, mikä helpottaa ja nopeuttaa arviointia siitä, mille koneelle uusi toiminnallisuus voidaan turvallisesti asentaa ilman suorituskykyongelmia. Tällainen näkyvyys tukee kapasiteettisuunnittelua ja auttaa ehkäisemään resurssipuutteista johtuvia haasteita.

Kolmas esimerkkitilanne liittyy virtuaalikoneen tallennustilan loppumiseen. Mikäli järjestelmä tuottaa hälytyksen levytilan vähyydestä, ylläpitäjä voi nopeasti tarkistaa taulukosta, kuinka paljon kyseisellä virtuaalikoneella on allokoitu tallennustilaa. Ylläpitäjä voi tämän jälkeen kirjautua suoraan oikeaan hallintajärjestelmään tai jopa suoraan kyseiselle virtuaalikoneelle vapauttamaan levytilaa esimerkiksi siivoamalla tarpeetonta dataa tai vaihtoehtoisesti laajentaa levytallennustilaa käytettävissä olevien resurssien puitteissa. Näin reagointi- ja korjaustoimet voidaan kohdistaa suoraan oikeaan koneeseen ja suorittaa tehokkaasti.

Projektin tuottama tiedon koonti ja selkeä saavutettavuus ovatkin osoittautuneet erittäin hyödyllisiksi IT-infrastruktuurin operatiivisessa hallinnassa, erityisesti kiireellisissä vikatilanne-, ongelmanselvitys- ja kapasiteettisuunnittelutapauksissa.

7.2 IT-alan kehitys ja automaatiotrendit

IT-alan toimintaympäristöissä on viime vuosina tapahtunut huomattavaa kehitystä kohti yhä automatisoidumpia, skaalautuvampia ja pilvipohjaisia ratkaisuja. Automatisointi on noussut keskeiseksi tavoitteeksi niin infrastruktuurin hallinnassa, palveluiden ylläpidossa kuin tietoturvassakin. Tietotekniikan resurssien tehokkaampi hyödyntäminen sekä manuaalisen työn vähentäminen ovat muodostuneet strategisesti tärkeiksi teemoiksi organisaatioille kaikilla toimialoilla.

Automatisoidut tiedonkeruu- ja raportointijärjestelmät, kuten tässä projektissa toteutettu ratkaisu, heijastavat modernien IT-organisaatioiden perustoimintatapoja. Pilvipalveluiden, kuten Google Drivessa sijaitsevien jaetun tiedon, hyödyntäminen helpottaa yhteistyötä sekä mahdollistaa tiedon jakamisen ja hallinnan ajasta ja paikasta riippumatta. Tämä tukee niin paikallista kuin hajautettua työskentelyä ja on osa DevOps-toimintamallin ja jatkuvan palveluntuotannon yleistymistä.

Lisäksi ketteryyttä, skaalautuvuutta ja automaatiota korostavat käytännöt näkyvät esimerkiksi infrastruktuuri palveluna -ratkaisuissa (IaaS), käyttöoikeuksien hallinnassa sekä resurssien elinkaaren automatisoinnissa. Ajantasainen ja keskitetysti saavutettava data tukee nopeaa päätöksentekoa, resurssien optimaalista käyttöä sekä aikakriittisiin ongelmatilanteisiin reagointia.
Tässä projektissa hyödynnetyt ratkaisut ja käytetyt teknologiat edustavat myös IT-alan ajankohtaisia kehityssuuntia: avoimen lähdekoodin työkalut, laajat ohjelmointirajapinnat, pilvipohjaiset jakelualustat sekä prosessien automatisointi ovat kaikki tekijöitä, jotka muodostavat kestävän pohjan nykyaikaiselle IT-infrastruktuurille.

Kokonaisuudessaan projekti osoittaa kuinka automaatio ja digitaaliset työkalut mahdollistavat uudenlaista tehokkuutta ja laatua arjen IT-hallintaan.

8 POHDINTA JA HENKILÖKOHTAINEN OPPIMINEN

Tässä luvussa on yhteenveto projektista. Projektia tarkastellaan tiivistelmän, saavutuksien ja lopullisen pohdinnan kautta. Henkilökohtaista kasvua ja ammatillista oppimista kuvataan teknisten taitojen, projektinhallinnan, dokumentoinnin sekä kommunikoinnin näkökulmista.

8.1 Projektin tiivistelmä

Tämän opinnäytetyön tavoitteena oli suunnitella ja toteuttaa automatisoitu ratkaisu VMware-virtuaaliympäristöjen tietojen keräämiseen ja raportointiin sekä tiedon keskitettyyn tallentamiseen Google Driveen. Projektissa kehitettiin Python-pohjainen skripti, joka hyödyntää vSphere API:a sekä Google Drive API:a, ja joka on ajastettu Linux-palvelimen crontab-toiminnolla säännöllisesti suoritettavaksi. Ratkaisun avulla organisaation virtuaalikoneiden keskeiset tiedot – kuten resurssikäyttö, verkko-osoitteet ja muut olennaiset VM-attribuutit – kerätään automaattisesti, jäsennellään selkeään taulukkomuotoon ja tallennetaan pilvipohjaiseen kansioon, jossa tiedot ovat helposti saavutettavissa kaikille ylläpidon sidosryhmille. Työn aikana panostettiin erityisesti järjestelmän luotettavuuteen, skaalautuvuuteen ja tietoturvaan sekä varmistettiin, että järjestelmä soveltuu erilaisiin infrastruktuurikokoihin ja toimintaympäristöihin.

Automatisoitu tiedonkeruu ja raportointi vähentävät manuaalointyötä, nopeuttavat ongelmanselvitystä sekä parantavat IT-infrastruktuurin dokumentaation ajantasaisuutta ja saavutettavuutta. Projekti osoitti, että monimutkaisiakin järjestelmähallinnan tehtäviä voidaan tehostaa ja yksinkertaistaa sujuvasti automatisoiduilla työnkuluilla.

8.2 Keskeiset saavutukset

Projektin merkittävimpiä saavutuksia ovat järjestelmän selkeä vaikutus IT-ympäristön ylläpidon tehokkuuden ja sujuvuuden parantamiseen. Automatisoitu tiedonkeruu on merkittävästi helpottanut ja nopeuttanut virtuaalikoneiden tilatietojen hakua, sillä ajantasainen ja yhdenmukainen data on aina saatavilla keskitetysti Google Driven taulukossa. Ylläpidon henkilöstö on pystynyt käyttämään tätä tietoa nopeasti erilaisissa virhe- ja ongelmatilanteissa, mikä on lyhentänyt vianmäärityksen ja korjausten kokonaisaikaa.

Ratkaisun ansiosta manuaalinen tiedonhakutyö on vähentynyt merkittävästi, ja useat erilliset järjestelmien selailuun liittyvät vaiheet ovat poistuneet työprosessista. Yhteen näkymään kootut tiedot helpottavat kommunikointia sekä tiedon jakamista tiimin jäsenten välillä. Järjestelmän käyttöönotto on myös parantanut datan luotettavuutta ja poistanut monia inhimilliseen virheeseen liittyviä riskejä.
Kokonaisuutena projekti on tuottanut konkreettista hyötyä arjen toimintaan ja osoittanut automatisoinnin tuoman arvon osana IT-infrastruktuurin hallintaa.

8.3 Lopulliset pohdinnat

Projektin toteutus vahvisti käsitystä siitä, että IT-infrastruktuurin hallinnan automaatio tarjoaa merkittäviä etuja käytännön työn tehostamisessa ja laatuun vaikuttamisessa. Virtuaaliympäristön tiedonkeruun ja raportoinnin automatisointi mahdollisti nopeamman reagoinnin poikkeamatilanteisiin sekä vähensi rutiininomaisen manuaalityön tarvetta, mikä näkyi suoraan ylläpidon ajansäästönä ja virhetilanteiden nopeampana selvittämisenä.

Työn aikana kävi ilmi, että automaattinen tiedonkeruu ei ainoastaan helpota ylläpitohenkilöstön arkea, vaan myös parantaa organisaation tiedonhallintaa ja dokumentaation yhtenäisyyttä. Automaattisesti päivittyvät taulukot tukevat päätöksentekoa ja mahdollistavat paremman läpinäkyvyyden koko IT-infrastruktuurissa.

Lisäksi projektin myötä korostui dokumentoinnin sekä järjestelmä- ja tietoturvan jatkuvan kehittämisen tärkeys, sillä automaattinen ratkaisu tulee sopeuttaa muuttuviin teknisiin vaatimuksiin ja ympäristöihin. Yhteistyö ylläpidon kanssa, käyttäjäpalautteen saaminen ja ratkaisun jatkokehitys mahdollistavat järjestelmän pitkäikäisen ja laajenevan hyödyn myös tulevaisuudessa.
Kokonaisuutena projekti osoitti, että hyvin suunniteltu ja toteutettu automaatioratkaisu voi muuttaa olennaisesti sitä, miten IT-toimintoja ylläpidetään ja kehitetään – tuoden lisää tehokkuutta, skaalautuvuutta ja parempaa palvelua organisaation sisällä.

8.4 Omat tekniset taidot

Projektin toteutus tarjosi merkittäviä mahdollisuuksia kehittää ja syventää omaa teknistä osaamista erityisesti automaation, ohjelmointirajapintojen (API) sekä pil-

vipalveluintegroinnin alueilla. Käytännön työskentely Python-ohjelmoinnin parissa vahvisti ymmärrystä niin ohjelmakomponenttien rakenteesta, kirjastoista kuin riippuvuuksien hallinnasta.
Projektin myötä karttui osaamista seuraavilla alueilla:

8.4.1 Python-ohjelmointi

Skriptin suunnittelu, toteutus ja testaus syvensivät taitoja Python-kielen soveltamisessa automaatiossa, käskyvirran hallinnassa ja virheenkäsittelyssä.
Rajapintaintegraatiot: Työskentely VMware vSphere API:n, pyvmomi-kirjaston sekä Google Drive API:n ja Sheets API:n kanssa avasi käytännön näkökulmia API-kutsuihin, dokumentaation tulkintaan ja yhteensopivuusongelmien ratkaisemiseen.

8.4.2 Tietojen automaattinen prosessointi

Taulukkorakenteiden hyödyntäminen toi kokemusta datan keruusta, käsittelystä sekä muotoilusta erilaisiin raportti- ja analyysitarpeisiin.
Autentikointi ja valtuutus: OAuth 2.0:n käyttö sekä todennustiedostojen ja käyttöoikeuksien hallinta paransivat ymmärrystä pilvipalveluihin liittyvistä tietoturvavaatimuksista ja tunnistautumisprosesseista.

8.4.3 Automaatio ja Linux-ympäristö

Crontabin ja muiden Linuxin ajastus- ja valvontatyökalujen käyttö syvensi kokonaiskuvaa palvelinympäristöjen automatisoinnin mahdollisuuksista ja rajoituksista.

Näiden teknisten taitojen kehittyminen on luonut pohjaa entistä monipuolisempiin ja vaativampiin automaatioratkaisuihin sekä helpottaa jatkossa uusien järjestelmien käyttöönottoa ja kehitystä.

8.5 Omat projektinhallintataidot

Projektin toteutus ja sen eri vaiheet kehittivät huomattavasti omaa projektinhallintaosaamista. Työskentely monivaiheisen ja useita ulkoisia rajapintoja hyödyntävän järjestelmän parissa vaati sekä suunnitelmallisuutta että kykyä hallita useita tehtäviä rinnakkain.
Projektin aikana opin erityisesti seuraavia projektinhallintataitoja:

8.5.1 Aikataulutus ja tehtävien priorisointi

Eri kehitysvaiheiden, kuten vaatimusmäärittelyn, toteutuksen, testauksen ja käyttöönoton, suunnittelu sekä aikataulutus auttoivat pitämään projektin etenemisen tavoitteellisena ja hallittuna.

8.5.2 Resurssien hallinta

Rajatun ajankäytön sekä käytössä olevien ohjelmisto- ja kehitysympäristöresurssien tehokas hyödyntäminen osoittautui tärkeäksi projektin onnistumiselle. Lisäksi ylläpidettävyyden ja jatkokehitettävyyden huomioiminen ratkaisuissa helpotti työn etenemistä.

8.5.3 Riskien tunnistaminen ja hallinta

Projektissa tuli eteen lähinnä teknisiä ongelmia, kuten API-yhteensopivuusongelmia ja käytettävien teknologioiden dokumentaatiopuutteita.

8.5.4 Dokumentointi

Kokonaisuuden hallinta vaati huolellista dokumentointia projektin jokaisessa vaiheessa. Selkeät muistiinpanot, ylläpito-ohjeet ja testausdokumentit ovat olleet hyödyksi kokonaisuuden ylläpidon ja tiedon siirron varmistamisessa.
Projektin läpivienti auttoi ymmärtämään, miten järjestelmällinen eteneminen, jatkuva oman työn seuraaminen ja aktiivinen riskien arviointi vaikuttavat sekä projektin sujuvuuteen että lopputuloksen laatuun.

8.6 Yhteistyö ja viestintä

Projektin toteutus tapahtui tiiminvanhimman antaman tehtävänannon perusteella, mutta vastuu suunnittelusta, toteutuksesta ja dokumentaatiosta oli pääosin minulla. Työskennellessäni itsenäisesti opin hallitsemaan kokonaisuutta, aikatauluttamaan omat tehtäväni sekä seuraamaan työn etenemistä järjestelmällisesti.

Projektin eri vaiheiden aikana konsultoin kollegoita erityisesti teknisissä kysymyksissä ja rajapintojen yhteensopivuuteen liittyvissä haasteissa. Asiantuntija-apu tiimiltä auttoi ratkaisemaan yksittäisiä ongelmakohtia sekä varmistamaan, että

käyttöönotetut ratkaisut noudattavat organisaation parhaita käytäntöjä ja toimintamalleja. Tiimiltään saadun palautteen ja keskustelujen pohjalta pystyin myös kehittämään ratkaisua entistä toimivammaksi käytännön tarpeiden mukaisesti.

Projektin valmistuttua esittelin työn lopputulokset, tarkoituksen ja toteutuksen ylläpitotiimille. Esittelyn avulla varmistin, että kaikki sidosryhmät ymmärtävät projektin merkityksen, toimintaperiaatteen ja hyödyt. Yhteisissä keskusteluissa voitiin jakaa kokemuksia, nostaa esiin kehitysideoita ja sopia ylläpitovastuut projektille jatkossa.

Kokonaisuutena projekti vahvisti viestintä- ja yhteistyötaitojani, opetti argumentoinnin ja tiedon selkeän jakamisen merkityksen sekä kehitti kykyä toimia itseohjautuvasti IT-kehitysprojektin eri vaiheissa.

LÄHTEET

Broadcom. (2024). *Python access to vSphere APIs.* **https://techdocs.broadcom.com/us/en/vmware-cis/vsphere/vsphere-sdks-tools/8-0/an-introduction-getting-started-with-vsphere-apis-and-sdks-8-0/getting-started-with-vsphere-apis-and-sdks/python-access-to-vsphere-apis.html**

DataCamp. (2024). *Automate Anything With Python: A Practical Guide to Python Automation.* **https://www.datacamp.com/tutorial/python-automation**

Google Cloud. (2024). *What is a virtual machine?* **https://cloud.google.com/learn/what-is-a-virtual-machine**

Google Developers Drive. (2024). Google Drive API documentation. https://developers.google.com/drive

Google Developers OAuth. (2024). OAuth 2.0 for Google APIs. https://developers.google.com/identity/protocols/oauth2

IBM. (2024). *What is virtualization?* **https://www.ibm.com/topics/virtualization**

Python Software Foundation. (2024). *About Python*. https://www.python.org/doc/essays/blurb/

LIITTEET

Liite 1. Skripti

1(10) jatkuu

```
from pyVim.connect import SmartConnect, Disconnect
from pyVmomi import vim
import ssl
import os
from google.oauth2.credentials import Credentials
from google_auth_oauthlib.flow import InstalledAppFlow
from google.auth.transport.requests import Request
from googleapiclient.discovery import build
from googleapiclient.http import MediaFileUpload
from docx import Document
from docx.shared import Inches
import pickle
import json
from dotenv import load_dotenv

load_dotenv()

# Disable SSL warnings (not recommended for production code)
ssl._create_default_https_context = ssl._create_unverified_context

# Google Drive API setup
SCOPES = ['https://www.googleapis.com/auth/drive', 'https://www.googleapis.com/auth/spread-
sheets']
FOLDER_ID = os.getenv('DRIVE_FOLDER_ID')
credentials_file_path = os.getenv('CREDENTIALS_FILE_PATH')
token_file_path = os.getenv('TOKEN_FILE_PATH')
drive_port = os.getenv('DRIVE_PORT')

# vSphere host connection parameters
user = os.getenv('USER')
internal_host = os.getenv('INTERNAL_HOST')
internal_password = os.getenv('INTERNAL_PASSWORD')
external_host = os.getenv('EXTERNAL_HOST')
external_password = os.getenv('EXTERNAL_PASSWORD')
redirect_uri = os.getenv('REDIRECT_URI')
vm_port = os.getenv('VM_PORT')
```

```
def connect_to_vcenter(host, user, password, port):

    service_instance = SmartConnect(host=host,                                    2(10)
                                    user=user,
                                    pwd=password,
                                    port=vm_port)
    return service_instance

def get_all_vms(content):

  # Get all VMs from vSphere
    vm_view = content.viewManager.CreateContainerView(content.rootFolder,
                                          [vim.VirtualMachine],
                                          True)
    obj = [vm for vm in vm_view.view]
    vm_view.Destroy()
    print("Virtual machines found:")
    return obj
    print(obj)

def service_account_login():
    """Log in to Google API using OAuth2 credentials and return the service object."""
    creds = None
    # The file token.pickle stores the user's access and refresh tokens and is
    # created automatically when the authorization flow completes for the first time.
    if os.path.exists(token_file_path):
        with open(token_file_path, 'rb') as token:
            creds = pickle.load(token)
    # If there are no (valid) credentials available, let the user log in.
    if not creds or not creds.valid:
        if creds and creds.expired and creds.refresh_token:
            creds.refresh(Request())
        else:
            flow = InstalledAppFlow.from_client_secrets_file(credentials_file_path, SCOPES, redi-
rect_uri=redirect_uri)
            creds = flow.run_local_server(port=drive_port)
         #Save the credentials for the next run.
        with open(token_file_path, 'wb') as token:
            pickle.dump(creds, token)
    drive_service = build('drive', 'v3', credentials=creds)
    sheets_service = build('sheets', 'v4', credentials=creds)
    return drive_service, sheets_service
```

```
def get_network_info(vm):                                                    3(10)
    # This function will extract network adapters and IP addresses from the VM
    network_info = []
    for device in vm.config.hardware.device:
        if isinstance(device, vim.vm.device.VirtualEthernetCard):
            ip_addresses = []
            # Searching for IP addresses in the guest.net property
            for net in vm.guest.net:
                if net.macAddress == device.macAddress and net.ipConfig is not None:
                    # Add a check here to ensure ipConfig is not None
                    for ipAddress in net.ipConfig.ipAddress:
                        # You might want to include only specific types of IP addresses
                        if ipAddress.ipAddress and not ipAddress.ipAddress.startswith('fe80'):  # Exclude
link-local IPv6 addresses
                            ip_addresses.append(ipAddress.ipAddress)
            network_info.append({
                'network': device.deviceInfo.summary,
                'ip_addresses': ip_addresses  # List of IP addresses associated with the network
adapter
            })
    return network_info

def get_host_ip(vm):
    # This function will attempt to get the IP address of the host system
    host = vm.runtime.host
    if host:
        for nic in host.config.network.vnic:
            if nic.portgroup == 'Management Network' or nic.portgroup == 'VMkernel':
                return nic.spec.ip.ipAddress
    return "Host IP not found"

def get_dns_name(vm):

    try:
        # Check if VMware Tools is installed and running to get accurate info
        if vm.guest.toolsRunningStatus == 'guestToolsRunning':
            dns_name = vm.guest.hostName
            if dns_name:
                return dns_name
            else:
                return "DNS name not available"
```

```
        else:
            return "VMware Tools not running"                                    4(10)
    except Exception as e:
        print(f"Error retrieving DNS name for VM: {str(e)}")
        return "Error"

def add_vm_info_to_sheet(sheets_service, spreadsheet_id, vms, sheet_title):
    # Create a new sheet in the existing spreadsheet
    batch_update_spreadsheet_request_body = {
        'requests': [{
            'addSheet': {
                'properties': {
                    'title': sheet_title,
                    'gridProperties': {
                        'rowCount': len(vms) + 1,
                        'columnCount': 7
                    }
                }
            }
        }]
    }
    response = sheets_service.spreadsheets().batchUpdate(
        spreadsheetId=spreadsheet_id,
        body=batch_update_spreadsheet_request_body).execute()

    # Get the ID of the newly created sheet
    new_sheet_id = response['replies'][0]['addSheet']['properties']['sheetId']

    # Prepare and insert data into the new sheet
    values = [['Host IP', 'VM Name', 'DNS Name', 'Resource Pool', 'Network Adapters and IP Addresses', 'CPU Count', 'Memory (MB)', 'Total Hard Disk Size (GB)', 'Power State', 'VM Note']]  # Header row
    for vm in vms:
        host_ip = get_host_ip(vm)
        vm_name = vm.summary.config.name
        dns_name = get_dns_name(vm)
        resource_pool = vm.resourcePool.name if vm.resourcePool else "No resource pool"
        power_state = vm.runtime.powerState
        network_info = get_network_info(vm)
        network_adapters_str = json.dumps(network_info)
        cpu_count = vm.summary.config.numCpu
        memory_size_mb = vm.summary.config.memorySizeMB
```

```
        total_hard_disk_size_gb = 0
        for device in vm.config.hardware.device:                                    5(10)
            if isinstance(device, vim.vm.device.VirtualDisk):
                hard_disk_size_gb = device.capacityInKB // (1024 * 1024)  # Convert from KB to GB
                total_hard_disk_size_gb += hard_disk_size_gb
        vm_note = vm.summary.config.annotation if vm.summary.config.annotation else "No note"

        values.append([host_ip, vm_name, dns_name, resource_pool, network_adapters_str,
cpu_count, memory_size_mb, total_hard_disk_size_gb, power_state, vm_note])

    # Insert data into the new sheet
    body = {
        'values': values
    }
    range_ = f"{sheet_title}!A1:J{len(vms) + 1}"
    sheets_service.spreadsheets().values().update(
        spreadsheetId=spreadsheet_id, range=range_,
        valueInputOption='RAW', body=body).execute()

    # Apply formatting to the sheet
    requests = [
        {
            'updateSheetProperties': {
                'properties': {
                    'sheetId': new_sheet_id,
                    'gridProperties': {
                        'frozenRowCount': 1
                    }
                },
                'fields': 'gridProperties.frozenRowCount'
            }
        },
        {
            'setBasicFilter': {
                'filter': {
                    'range': {
                        'sheetId': new_sheet_id,
                        'startRowIndex': 0,
                        'startColumnIndex': 0,
                        'endColumnIndex': 10
                    }
                }
```

```
            }
        },                                                                    6(10)
        {
            "autoResizeDimensions": {
                "dimensions": {
                    "sheetId": new_sheet_id,
                    "dimension": "COLUMNS",
                    "startIndex": 0,
                    "endIndex": 10
                }
            }
        }
    ]

    # Execute the batchUpdate request to apply styles
    batch_update_request_body = {
        'requests': requests
    }
    sheets_service.spreadsheets().batchUpdate(
        spreadsheetId=spreadsheet_id, body=batch_update_request_body).execute()

    # Return the ID of the new sheet
    return new_sheet_id

def create_and_upload_sheet(sheets_service, drive_service, vms, folder_id, sheet_title):
    # Create a new Google Sheets spreadsheet
    spreadsheet = {
        'properties': {
            'title': 'VM Information'
        },
        'sheets': [{
            'properties': {
                'title': 'Internal VM Details',
                'gridProperties': {
                    'rowCount': len(vms) + 1,  # Number of VMs plus header row
                    'columnCount': 2
                }
            }
        }]
    }
    spreadsheet = sheets_service.spreadsheets().create(body=spreadsheet,
                                        fields='spreadsheetId,sheets.properties.sheetId').execute()
```

```
    spreadsheet_id = spreadsheet.get('spreadsheetId')
    first_sheet_id = spreadsheet['sheets'][0]['properties']['sheetId']                    7(10)

    # Prepare the data to be inserted into the spreadsheet
    values = [['Host IP', 'VM Name', 'DNS Name', 'Resource Pool', 'Network Adapters and IP Addresses', 'CPU Count', 'Memory (MB)', 'Total Hard Disk Size (GB)', 'Power State', 'VM Note']]  # Header row
    for vm in vms:
        host_ip = get_host_ip(vm)
        vm_name = vm.summary.config.name
        dns_name = get_dns_name(vm)
        resource_pool = vm.resourcePool.name if vm.resourcePool else "No resource pool"
        power_state = vm.runtime.powerState
        network_info = get_network_info(vm)
        network_adapters_str = json.dumps(network_info)
        cpu_count = vm.summary.config.numCpu
        memory_size_mb = vm.summary.config.memorySizeMB
        total_hard_disk_size_gb = 0
        for device in vm.config.hardware.device:
            if isinstance(device, vim.vm.device.VirtualDisk):
                hard_disk_size_gb = device.capacityInKB // (1024 * 1024)  # Convert from KB to GB
                total_hard_disk_size_gb += hard_disk_size_gb
        vm_note = vm.summary.config.annotation if vm.summary.config.annotation else "No note"

        values.append([host_ip, vm_name, dns_name, resource_pool, network_adapters_str, cpu_count, memory_size_mb, total_hard_disk_size_gb, power_state, vm_note])

    # Insert data into the spreadsheet
    body = {
        'values': values
    }
    range_ = 'Internal VM Details!A1:J' + str(len(vms) + 1)  # Adjust range to number of attributes
    sheets_service.spreadsheets().values().update(
        spreadsheetId=spreadsheet_id, range=range_,
        valueInputOption='RAW', body=body).execute()

    # Define the requests for freezing the first row and enabling filtering
    requests = [
        {
            # Freeze the first row
            'updateSheetProperties': {
                'properties': {
```

```
            'sheetId': first_sheet_id,
            'gridProperties': {                                                    8(10)
                'frozenRowCount': 1  # Number of rows to freeze
            }
        },
        'fields': 'gridProperties.frozenRowCount'
    }
  },


  {
    # Set basic filter on all columns
    'setBasicFilter': {
        'filter': {
            'range': {
                'sheetId': first_sheet_id,
                'startRowIndex': 0,
                'startColumnIndex': 0,
                'endColumnIndex': 10
            }
        }
    }
  },

  {
  "autoResizeDimensions": {
    "dimensions": {
        "sheetId": first_sheet_id,
        "dimension": "COLUMNS",
        "startIndex": 0,
        "endIndex": 10
            }
        }
    }
]

# Execute the batchUpdate request
batch_update_request_body = {
  'requests': requests
}

response = sheets_service.spreadsheets().batchUpdate(
```

```
        spreadsheetId=spreadsheet_id, body=batch_update_request_body).execute()

    # Move the spreadsheet to the desired folder in Drive                    9(10)
    # First, retrieve the file's parents to remove
    file = drive_service.files().get(fileId=spreadsheet_id,
                                     fields='parents').execute()
    previous_parents = ",".join(file.get('parents'))
    # Move the file to the new folder
    file = drive_service.files().update(fileId=spreadsheet_id,
                                        addParents=folder_id,
                                        removeParents=previous_parents,
                                        fields='id, parents').execute()

    # Return the ID of the new spreadsheet
    return spreadsheet_id

def main():
    # Connect to internal vCenter
    internal_service_instance = connect_to_vcenter(internal_host, user, internal_password,
vm_port)  # Capture the ServiceInstance object here
    internal_content = internal_service_instance.RetrieveContent()
    print("Connected to internal vCenter")

    # Fetch all internal VMs
    internal_vms = get_all_vms(internal_content)

    # Log in to Google Drive
    drive_service, sheets_service = service_account_login()  # Unpack the returned tuple

    # Create and upload a Google Sheets spreadsheet to Google Drive with VM information
    spreadsheet_id = create_and_upload_sheet(sheets_service, drive_service, internal_vms,
FOLDER_ID, sheet_title='Internal VM Details')
    print(f"Spreadsheet created with ID: {spreadsheet_id}")

    # Disconnect from internal vCenter
    Disconnect(internal_service_instance)
    print("Internal vCenter connection disconnected")

    # Connect to external vCenter
    external_service_instance = connect_to_vcenter(external_host, user, external_password,
vm_port)  # Capture the ServiceInstance object here
    external_content = external_service_instance.RetrieveContent()          10(10)
```

```
    print("Connected to external vCenter")

    # Fetch all external VMs
    external_vms = get_all_vms(external_content)

    # Add a new sheet and populate it with external VM information from the second vCenter
    add_vm_info_to_sheet(sheets_service, spreadsheet_id, external_vms, sheet_title='External
VM Details')
    print("Added External VM details to the spreadsheet")

    # Disconnect from external vCenter
    Disconnect(external_service_instance)
    print("External vCenter connection disconnected")

if __name__ == "__main__":
    main()
```

Liite 2. Dotenv-esimerkki

```
DRIVE_FOLDER_ID = <>
CREDENTIALS_FILE_PATH = <>
TOKEN_FILE_PATH = <>
DRIVE_PORT = <>
USER = <>
INTERNAL_HOST = <>
INTERNAL_PASSWORD = <>
EXTERNAL_HOST = <>
EXTERNAL_PASSWORD = <>
REDIRECT_URI = <>
VM_PORT = <>
```