VIŠJA STROKOVNA ŠOLA ACADEMIA

MARIBOR

AVTOMATIZACIJA SISTEMSKE ADMINISTRACIJE S POMOČJO SODOBNIH ORODIJ IN SKRIPTIRANJA

Kandidat: Iztok Hladen

Vrsta študija: višješolski strokovni študij

Študijski program: Informatika

Mentor – predavatelj: mag. Dušan Brglez

Mentor v podjetju: Matjaž Žnidarič, inž. inf.

Lektorica: Jasmina Vajda Vrhunec, prof. slov.

Maribor, 2025

IZJAVA O AVTORSTVU DIPLOMSKEGA DELA

Podpisani Iztok Hladen sem avtor diplomskega dela z naslovom Avtomatizacija sistemske administracije s pomočjo sodobnih orodij in skriptiranja, ki sem ga napisal pod mentorstvom mag. Dušana Brgleza.

S svojim podpisom zagotavljam, da:

- je predloženo delo izključno rezultat mojega dela,
- sem poskrbel, da so dela in mnenja drugih avtorjev, ki jih uporabljam v predloženi nalogi, navedena oz. citirana skladno s pravili Višje strokovne šole Academia Maribor,
- se zavedam, da je plagiatorstvo – predstavljanje tujih del oz. misli kot moje lastne – kaznivo po Zakonu o avtorski in sorodnih pravicah (Uradni list RS, št. 16/07 – uradno prečiščeno besedilo, 68/08, 110/13, 56/15 in 63/16 – ZKUASP); prekršek pa podleže tudi ukrepom Višje strokovne šole Academia Maribor skladno z njenimi pravili,
- skladno z 32.a členom ZASP dovoljujem Višji strokovni šoli Academia Maribor objavo diplomskega dela na spletnem portalu šole.

Križevci pri Ljutomeru, junij 2025 Podpis študenta: Iztok Hladen

ZAHVALA

Zahvaljujem se mentorju mag. Dušanu Brglezu za strokovno pomoč, usmeritve in koristne nasvete skozi celoten proces nastajanja diplomskega dela.

Prav tako se zahvaljujem mentorju v podjetju g. Matjažu Žnidariču za podporo in pomoč pri praktični uporabi orodij, potrebnih za nastanek diplomskega dela.

Največjo zahvalo namenjam družini za neomajno podporo, razumevanje in spodbudo, ne le med pisanjem diplomskega dela, temveč skozi celoten študij.

POVZETEK

V sodobnih okoljih informacijske tehnologije (IT) predstavlja sistemska administracija temelj za zagotavljanje stabilnega, varnega in učinkovitega delovanja IT-sistemov. Cilj diplomskega dela je bil ovrednotiti vpliv avtomatizacije ključnih nalog sistemske administracije na učinkovitost, zanesljivost, varnost in odzivnost IT-sistemov.

Diplomsko delo se začne z zgodovinskim pregledom računalništva in opredelitvijo vloge sistemskega administratorja. Sledi pregled izbranih orodij, ki podpirajo avtomatizacijo in napredno analitiko – od osnovnih skriptnih jezikov, kot so Shell, PowerShell in Python, do specializiranih orodij, kot sta Ansible za upravljanje konfiguracij in CheckMK za nadzor sistemov. Predstavljena so tudi nekatera napredna orodja, kot je Darktrace, ki izkorišča umetno inteligenco in strojno učenje za izboljšano zaznavanje anomalij ter podporo pri odločanju tako v sistemski administraciji kot v kibernetski varnosti. Teoretično je obravnavan tudi pomen avtomatizacije za varnostno skladnost, zlasti v kontekstu upravljanja uporabniških računov. Osrednji del diplomskega dela se nato osredotoča na poglobljeno implementacijo avtomatizacije specifičnih sistemskih opravil, kjer s praktičnimi primeri preverjamo nekatere zastavljene hipoteze. V tem sklopu smo razvili tudi lastno PowerShell skripto AD-Toolkit, ki skozi interaktivni meni ponuja nabor orodij za upravljanje uporabniških računov v okolju Active Directory. Prav tako smo izvedli avtomatizacijo varnostnega kopiranja podatkov in obnovitve sistema z orodjem Proxmox VE ter avtomatizirano nameščanje programskih posodobitev z orodjem Ansible.

Rezultati potrjujejo, da avtomatizacija sistemskih opravil zmanjšuje tveganje izgube podatkov in število varnostnih ranljivosti, izboljšuje odzivni čas na sistemske težave, optimizira porabo diskovnega prostora in skrajša čas obnovitve sistemov po napakah. Čeprav so nekatere naprednejše tematike, kot je polna implementacija rešitev na osnovi umetne inteligence zaradi praktičnih omejitev, obdelane pretežno skozi teoretični del, praktični preizkusi in razvoj lastnega orodja jasno kažejo, da je avtomatizacija nepogrešljiv element za doseganje visoke učinkovitosti zanesljivosti, varnosti in odzivnosti sodobnih IT-sistemov.

Ključne besede: *avtomatizacija, sistemska administracija, PowerShell, Ansible, varnost informacijskih sistemov.*

ABSTRACT

Automation of System Administration Using Modern Tools and Scripting

In modern IT environments, system administration forms the foundation for ensuring the stable, secure and efficient operation of IT systems. The aim of this diploma thesis was to evaluate the impact of automating key system administration tasks on the efficiency, reliability, security, and responsiveness of IT systems.

The diploma thesis starts with a historical overview of computing and a definition of the system administrator's role. This is followed by a review of the selected tools that support automation and advanced analytics – from basic scripting languages such as Shell, PowerShell, and Python, to specialized tools like Ansible for configuration management and CheckMK for system monitoring. Some advanced tools are also presented, such as Darktrace, which utilizes artificial intelligence and machine learning for improved anomaly detection and supports decision making in both system administration and cybersecurity. The importance of automation for security compliance, particularly in the context of user account management, is also discussed theoretically. The core part of the diploma thesis then focuses on the implementation of automating specific system tasks, where practical examples are used to test some of the hypotheses. In this section, we also developed our own PowerShell script, AD-Toolkit, which provides a set of tools for managing user accounts in Active Directory via an interactive menu. We further implemented automation of data backup and system recovery using Proxmox VE, and automated software update installation with Ansible.

The results confirm that automating system tasks reduces the risk of data loss, decreases the number of security vulnerabilities, improves response time to system errors, optimizes disk space usage, and shortens system recovery time after failures. Although some more advanced topics, such as the full implementation of artificial intelligence-based solutions, were addressed primarily in the theoretical part due to practical constraints, the practical tests and the development of a custom tool clearly indicate that automation is an important element for achieving high efficiency, reliability, security, and responsiveness in modern IT systems.

Keywords: *automation, system administration, PowerShell, Ansible, IT System Security*

KAZALO VSEBINE

KAZALO SLIK

1 UVOD

1.1 Opis področja in opredelitev problema

V današnjem svetu je informacijska tehnologija (IT) postala hrbtenica poslovanja, zato je vloga sistemske administracije ključnega pomena pri zagotavljanju nemotenega delovanja in zanesljivosti IT-sistemov. Naloge sistemskih administratorjev zajemajo širok spekter, v katerega vključujemo upravljanje uporabnikov in strežnikov, nameščanje in konfiguracijo programske opreme, postavljanje in vzdrževanje omrežij ter zagotavljanje varnosti sistemov. S stalnim napredkom tehnologije in vedno večjo kompleksnostjo IT-okolij pa se sistemski administratorji soočajo z vedno večjimi izzivi. V sodobnem IT-svetu so tradicionalne metode in pristopi pogosto zamudni in podvrženi napakam. Kadar je treba upravljati na stotine ali celo na tisoče strežnikov in naprav, postane ročna sistemska administracija zamudna, neučinkovita in praktično nemogoča. Taka neučinkovitost povzroči izgubo časa in virov, prav tako pa lahko povzroči resne težave, kot so izpadi raznih sistemov, varnostne ranljivosti in izguba podatkov. V zadnjem času se tudi na področju sistemske administracije uveljavlja uporaba umetne inteligence, kar sistemskim administratorjem lahko tako olajša kot do določene mere tudi oteži delo.

Namen diplomskega dela je raziskovati razne možnosti avtomatizacije sistemske administracije z namenom izboljšanja učinkovitosti, zmanjšanja stroškov in hkrati povečanja zanesljivosti IT-sistemov. Avtomatizacija predstavlja rešitev za raznorazne izzive, s katerimi se sistemski administratorji soočajo na dnevni bazi, saj nam omogoča avtomatizacijo ponavljajočih se nalog, standardizacijo konfiguracij in avtomatizirano spremljanje sistemov. S pomočjo avtomatizacije zmanjšamo potrebo po ročnem posredovanju, ob pravilni konfiguraciji pa s tem zmanjšamo možnost človeških napak in prav tako sprostimo časovne vire sistemskih administratorjev, zaradi česar se lahko posvetijo pomembnejšim nalogam.

V diplomskem delu se bomo osredotočili na praktično implementacijo avtomatizacije nekaterih ključnih nalog sistemske administracije. Za to bomo primarno uporabili skriptni jezik PowerShell. Če je za specifične naloge bolj smiselno uporabiti okolje Linux, se bomo posluževali Shell skript, omenili pa bomo tudi uporabo skript, zapisanih v programskem jeziku Python, ter naprednejša in zmogljivejša orodja, kot je Ansible.

1.2 Namen, cilji in osnovne trditve

Namen diplomskega dela je raziskati in ovrednotiti vpliv avtomatizacije ključnih nalog sistemske administracije na učinkovitost, zanesljivost, varnost in odzivnost IT-sistemov. Poudarek bo na uporabi skriptnega jezika PowerShell (predvsem v okolju Microsoft Windows), dopolnjenega s Shell skriptami (v okoljih Linux) in Python skriptami. Dodatno bomo še raziskali možnosti uporabe umetne inteligence za izboljšanje učinkovitosti avtomatizacije.

Osredotočili se bomo predvsem na avtomatizacijo varnostnega kopiranja podatkov, avtomatizacijo posodabljanja sistemov, avtomatizacijo čiščenja začasnih datotek, avtomatizacijo spremljanja sistemskih virov in uporabo umetne inteligence pri avtomatizaciji.

1.3 Predpostavke in omejitve

Diplomsko delo je zastavljeno na način, da imamo relativno enostaven dostop do potrebne tehnologije – z nekaj izjemami. Za izvedbo praktičnega dela projektnega dela je zahtevana določena mera osnovnega znanja sistemske administracije, kot je poznavanje ukazne vrstice tako v okoljih Windows kot Linux. Čeprav je dostop do tehnologije relativno enostaven, pa morda vidimo težavo v omejenosti potrebnih računalniških virov in posledično v omejenem testnem okolju. Prav tako zaradi informacij zaupne narave v diplomskem delu ne moremo izvesti celotnega praktičnega dela v okolju podjetja, zato se bomo omejili na lokalno postavljene strežnike in virtualne stroje.

1.4 Uporabljene raziskovalne metode

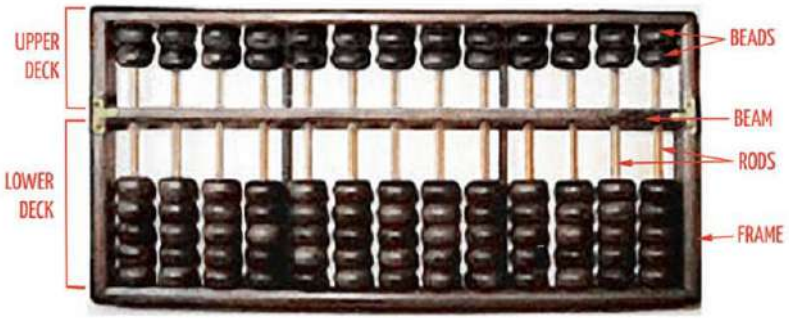
V diplomskem delu bomo uporabili kombinacijo empiričnega in praktično-aplikativnega pristopa. V raziskavi se bomo opirali na že obstoječe znanje in izkušnje strokovnjakov, ki so opisani v strokovni literaturi, hkrati pa bomo ustvarili uporabne skripte za avtomatizacijo in preverili, kako dobro se te obnesejo v praksi. Na tak način bo raziskava imela praktično vrednost in bo hkrati tudi dobro utemeljena s teorijo, kar bo pomagalo pri lažjem razumevanju namena avtomatizacije v sistemski administraciji.

2 ORODJA ZA AVTOMATIZACIJO SISTEMSKE ADMINISTRACIJE

2.1 Zgodovina avtomatizacije

2.1.1 Zgodovina računalnikov

Zgodovina razvoja računalnikov sega v davno preteklost, saj so si ljudje pri štetju pomagali z raznoraznimi pripomočki. Dobrih 2500 let pred našim štetjem se je pojavil abakus – naprava, ki je s pomočjo v okvir vpetih kroglic omogočala hitro in enostavno seštevanje, odštevanje in nekatere ostale računske operacije, s tem pa je kar precej poenostavila trgovanje. (Anželj in drugi, 2025, str. 212)



Slika 1: Abakus

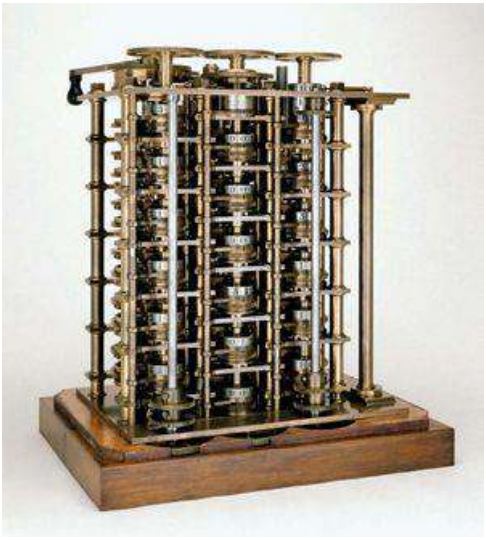
Vir: https://www.ecb.torontomu.ca/~elf/abacus/intro.html

Leta 1642 je Blaise Pascal naredil prvi mehanični kalkulator, ki ga imenujemo Pascaline. Deloval je na principu zobatih koles z desetimi zobmi ter mehanizma za prenos enote na naslednje kolo, omogočal pa je seštevanje in odštevanje. (Anželj in drugi, 2025, str. 213) Podobno delovanje je bilo mogoče opaziti v avtomobilih, saj so v preteklosti avtomobili imeli analogne števce kilometrov, ki so delovali na dokaj podoben princip.

Slika 2: Kalkulator Pascaline

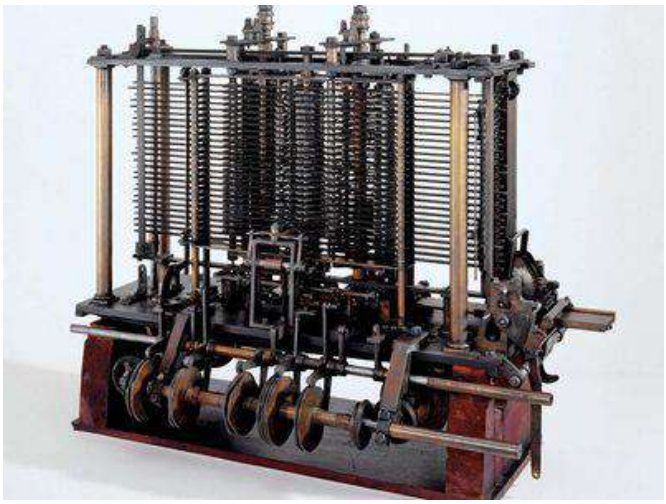
Vir: https://kids.britannica.com/students/article/Pascaline/332603/media?assemblyId=116010

Ker pa so ljudje vedno bolj stremeli k temu, da svoja dnevna opravila avtomatizirajo, se je v 19. stoletju pojavil nov izum. Charles Babbage je namreč izumil prvi avtomatični mehanski kalkulator – poimenovan diferenčni stroj. (Anželj in drugi, 2025, str. 214) Ta stroj je bil zmožen računati vrednosti s pomočjo seštevanja, kar pomeni, da je bil relativno omejen. Kasneje je Babbage zasnoval tako imenovani analitični stroj, ki je bil zmožen poleg seštevanja tudi odštevanja, množenja in deljenja, možno pa ga je bilo tudi programirati. Leta 1843 je Ada Lovelace v svojih prevodih člankov italijanskega matematika in inženirja Luigija Federica Menabreaja zasnovala prvi računalniški program za analitični stroj, ki bi lahko izračunaval Bernoullijeva števila. (Charles Babbage, 2025)



Slika 3: Diferenčni stroj

Vir: https://cdn.britannica.com/10/23610-050-6E34CF6B/portion-Difference-Engine-Charles-Babbage-logarithm-tables-1832.jpg

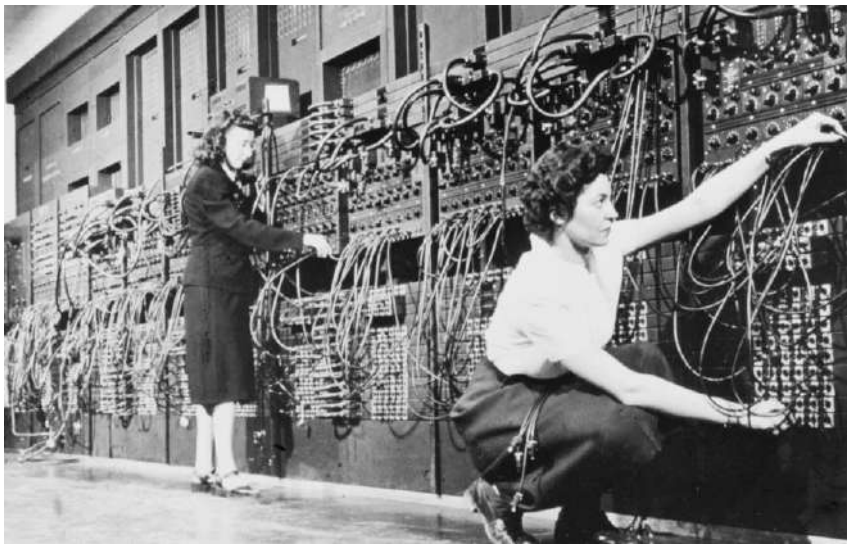


Slika 4: Analitični stroj

Vir: https://cdn.britannica.com/31/172531-050-E009D42C/portion-Charles-Babbage-Analytical-Engine-death-mill-1871.jpg

Skok v leto 1936 nam prinese novo teoretično zasnovo sodobnega računalnika, ki si ga je zamislil Alan Turing. Ta naj bi imel neskončen trak, katerega glava bi imela možnost določenih operacij, in kontrolni mehanizem. Trak je razdeljen v kvadrate, ki lahko vsebujejo informacije ali pa ne, glava pa ima možnost premikanja, branja in zapisovanja ter tudi brisanja iz kvadratov na traku. (Turing Machine, 2025) To je kasneje postala osnova za vse nadaljnje digitalne računalnike, ki delujejo po Neumannovem principu. (Anželj in drugi, 2025, str. 216)

Pri zgodovini razvoja računalnikov je pomembno omeniti tudi računalnik Eniac, zasnovan leta 1943 in dokončan leta 1945. Zasnovan je bil za izračunavanje vrednosti v tabeli dosega artilerije. Navodila so bila programirana s pomočjo strojnih jezikov, kar je pomenilo, da je stroj lahko deloval z veliko hitrostjo. (Eniac, 2025) To leto predstavlja tudi nek mejnik, saj od tega leta naprej ljudje postanejo operaterji oziroma programerji, računalnik pa je le še poimenovanje za stroj. (Anželj in drugi, 2025, str. 217)



Slika 5: Eniac

Vir: https://cdn.sanity.io/images/i2z87pbo/production/a9132a54f148d9eef366dbf5f7a8cb5c25603971-2500x1597.webp

V letih, ki so sledila, ločimo pet generacij elektronskih računalnikov. Prva generacija, v katero spada tudi računalnik Eniac, temelji na tehnologiji elektronk, uporablja strojni jezik in luknjaste kartice, programira pa se s premikanjem stikal. Druga generacija temelji na tehnologiji tranzistorjev, uporablja luknjaste kartice, programira pa se s simbolnimi jeziki. Tretja generacija temelji na tehnologiji čipov oziroma integriranih vezij, kot vhodno in izhodno napravo uporablja tipkovnico in monitor, programira pa se s pomočjo postopkovnih in opisnih programskih jezikov. Prav tako imajo računalniki že nameščene operacijske sisteme. Četrta generacija temelji na tehnologiji mikroprocesorjev, kar pomeni, da je na eno silicijevo ploščico postavljenih ogromno čipov. Kot vhodne in izhodne naprave se uporabljajo miška, tipkovnica, mikrofon, fotoaparat, monitor, tiskalnik, zvočniki. Programira se večinoma v višjenivojskih programskih jezikih. Peta generacija, prihodnost računalnikov, pa bo po vsej verjetnosti vsebovala kvantne računalnike ali pa skupine računalnikov – oblak (angl. cloud). (Anželj in drugi, 2025, str. 218–220)

2.1.2 Kaj je sistemska administracija

Sistemska administracija je zelo širok pojem, neka strnjena definicija pa bi bila, da je to disciplina upravljanja informacijskih sistemov. V to so vključeni postavitev, konfiguracija in vzdrževanje, saj tako poskrbimo za zanesljivo delovanje računalniških sistemov in omrežij.

Sistemski administratorji imajo zelo širok spekter nalog, ki so ključnega pomena za nemoteno delovanje podjetij in organizacij. Odvisno od velikosti in kompleksnosti organizacije lahko opravljajo različne naloge, vendar pa v večji meri med te naloge spadajo načrtovanje, konfiguracija, odpravljanje težav in vzdrževanje kompleksnih računalniških sistemov, ki so sestavljeni iz več komponent (na primer sistemi za upravljanje baz podatkov in spletni strežniki) in več strežnikov, ki so razporejeni po več omrežjih in različnih platformah operacijskih sistemov. (Barrett in drugi, 2004) Prav tako pa med naloge sistemskih administratorjev spadajo tudi zagotavljanje varnosti sistemov, pomoč končnim uporabnikom, pripravljanje dokumentacije in ne nazadnje tudi pripravljanje avtomatizacije za določene naloge. (What Does a Systems Administrator Do?, 2025) Ravno pri tej zadnji nalogi sistemskih administratorjev – avtomatizaciji – se pogosto pojavljajo razni miti, kot na primer, da je avtomatizacija nevarna in da se raje poslužujejo ročnega upravljanja nalog, saj je s tem zagotovljena večja kakovost. Seveda ta trditev ne drži, saj je pri vsaki nalogi, ki vsebuje človeški dejavnik, možno, da pride do težave. Naloga sistemskega administratorja je, da avtomatizacijo preveri v testnem okolju, s tem pa zagotovi čim boljše delovanje v živem okolju. (Limoncelli, Hogan, & Chalup, 2017, str. 74)

Kot že omenjeno, imajo sistemski administratorji širok spekter nalog, zato jih lahko ločimo glede na področje, s katerim se primarno ukvarjajo. Tako lahko imamo sistemske administratorje za omrežja, administratorje baz podatkov, administratorje sistemov v oblaku in podobno. (What Does a Systems Administrator Do?, 2025)

Prihodnost sistemske administracije je po eni strani polna izzivov, po drugi strani pa polna novih priložnosti. Predvidevamo, da bodo področja, kot so avtomatizacija, umetna inteligenca, računalništvo v oblaku in splošna kibernetska varnost, močno vplivala na razvoj nalog sistemskih administratorjev.

Da bi lažje razumeli delo sistemskih administratorjev, je treba razumeti razvoj in napredek sistemov in različnih arhitektur. Če na kratko povzamemo vrste teh sistemov:

1. Mainframe je neke vrste glavni računalnik, ki je ključen za obdelavo ogromne količine podatkov in izvedbo ključnih operacij. Sistemski administrator za tako arhitekturo je odgovoren za vzdrževanje, upravljanje in varnost takega sistema. Njegovo delo vključuje nameščanje razne programske opreme, nadzorovanje delovanja sistema in tudi skrb za varnost sistema. Kot zanimivost, mainframe sega v davno leto 1937 – tako imenovani IBM Automatic Sequence Controlled Calculator ali Harvard Mark I je bil prvi računalnik take

arhitekture. Prednosti take arhitekture so predvsem v zanesljivosti in dostopnosti. (Susnjara & Smalley, 2025)

2. Na drugi strani sistemske administracije pa se je v 70. letih pojavil UNIX. Ti sistemi so imeli pomembno vlogo pri razvoju računalništva, saj so neposredno vplivali na razvoj marsikaterega modernega operacijskega sistema. Sistemski administratorji morajo do določene mere poznati administracijo UNIX, saj se ta arhitektura oziroma neki derivati te arhitekture pogosto pojavljajo še dandanes. Delo administratorja tako vključuje vse od nameščanja in konfiguracij do upravljanja uporabnikov, nadzorovanja sistema ter tudi skriptiranja in avtomatizacije. (unix.org, 2025)
3. V kontekstu sistemske administracije je pomembno omeniti še Microsoftovo programsko opremo, ki ne temelji na arhitekturi UNIX, vendar je kljub temu z vidika sistemske administracije pomemben del vsakdanjika sistemskih administratorjev.

Dandanes je realnost, da se v sodobnem IT-okolju sistemski administratorji srečujejo z zelo heterogenim delom, saj upravljanje vseh teh raznovrstnih sistemov in arhitektur zahteva širok spekter znanj in spretnosti.

2.2 Pregled orodij

2.2.1 Shell skripte

Shell skripte, običajno uporabljene v sistemih UNIX, so skripte, ki jih sistemski administratorji tipično uporabljajo za lažje, ponavljajoče se naloge. V večini sistemov je privzeta »lupina« ali shell tako imenovani **bash** (angl. Bourne-again shell), določeni sistemi pa še vedno uporabljajo **sh** ali pa **ksh**. (Nemeth, Snyder, Hein, & Whaley, 2010, str. 29, 30) Kot že omenjeno, je **bash** odličen za relativno enostavne skripte, ki sistemskemu administratorju avtomatizirajo delo, ki bi ga običajno moral izvajati ročno v ukazni vrstici.

V bash skripti v prvi vrstici sistemu povemo, da gre za skripto. To storimo z zapisom *#!/bin/bash*. S tem deklariramo, da bo datoteko interpretirala bash lupina, ki se nahaja v mapi */bin*. Za komentarje v skripti uporabljamo simbol »#«. (Nemeth, Snyder, Hein, & Whaley, 2010, str. 37)

Enostaven primer bash skripte bi lahko bil, ko imamo kot sistemski administrator nalogo, da uredimo avtomatizacijo brisanja vsebine mape in njenih podmap, vendar hkrati ohranjamo strukturo samih map. V ta namen bomo kreirali skripto z imenom brisanje_datotek.sh.

```
#!/bin/bash
#
# Brisanje datotek
# Avtor: Iztok Hladen
# Datum: 20.02.2025
#
# Nastavi mapo, v kateri se bodo brisale datoteke.
#

MAPA_ZA_BRISANJE="/home/iztokh/Documents/scanners"

# Preverjanje, če mapa obstaja

if [ ! -d "$MAPA_ZA_BRISANJE" ]; then
  echo "Mapa '$MAPA_ZA_BRISANJE' ne obstaja."
  exit 1
fi

# Iskanje in brisanje vseh datotek v mapi

find "$MAPA_ZA_BRISANJE" -depth -type f -delete

echo "Brisanje datotek v mapi '$MAPA_ZA_BRISANJE' in podmapah je končano."

exit 0
```

Struktura omenjene skripte je zelo enostavna: najprej določi, da se bo skripta izvajala z bash lupino, sledi nekaj zakomentiranih vrstic, v katerih so zapisani glavna funkcija skripte ter tudi avtor in datum. Sledita definicija spremenljivke lokacije mape, v kateri bo skripta izvajala brisanje, nato pa še hitro preverjanje, ali zapisana lokacija sploh obstaja. Če ta ne obstaja, se izpiše sporočilo o napaki, skripta pa se zaključi. Če lokacija obstaja, pa bo skripta poiskala v mapi, ki je bila določena s spremenljivko *MAPA_ZA_BRISANJE*, vse datoteke in jih tudi izbrisala. Skripti smo določili iskanje le datotek z uporabo *-type f*. Skripta se zaključi s sporočilom, da je bilo brisanje datotek v vnaprej določeni mapi in podmapah končano.

Če želimo, da se omenjena skripta izvaja samodejno v določenih intervalih, lahko za to uporabimo Cron. Za namen tega moramo urediti crontab, v katerega moramo dodati vrstico, v kateri definiramo čas izvajanja in lokacijo skripte. Če bi na primer želeli, da se skripta izvaja vsako nedeljo ob enih zjutraj, bi ta vrstica izgledala tako:

0 1 * * 0 /home/iztokh/Documents/brisanje_datotek.sh

V prvem znaku definiramo minuto, v drugem uro, tretji in četrti sta dan v mesecu in mesec, z * določimo, da se izvaja vsak dan v mesecu in vsak mesec v letu, z zadnjim znakom pa definiramo dan v tednu. Sledi lokacija, kjer se skripta nahaja.

2.2.2 PowerShell

PowerShell je zmogljivo Microsoftovo orodje, ki ga sistemski administratorji uporabljajo za poenostavitev sistemskih opravil in avtomatizacijo. (What is PowerShell?, 2025) Temelji na ogrodju .NET in za razliko od tradicionalnega ukaznega poziva (command prompt) uporablja objekte in ne le besedilne nize. (PowerShell nadomešča ukazni poziv, 2025) Ukazi v PowerShellu so imenovani »*cmdlet*« in so tako rekoč majhni programi za izvajanje specifičnih nalog. (Chapter 1 - Getting started with PowerShell, 2025) Nekaj primerov teh cmdletov:

- *Get-ChildItem*, ki nam na enostaven način izpiše vsebino specifične mape;
- *Get-Process*, ki nam brez uporabe dodatnih parametrov izpiše seznam vseh aktivnih procesov;
- *Get-Service*, ki nam brez uporabe dodatnih parametrov izpiše seznam vseh storitev v sistemu.

PowerShell prav tako podpira skriptiranje, kar sistemskim administratorjem omogoča avtomatizacijo kompleksnejših nalog in tudi ustvarjanje lastnih orodij.

Preprost primer tega bi lahko bil preverjanje odprtih portov na želenih IP-naslovih:

```
#
# Preverjanje portov po IP naslovih
#
# Avtor: Iztok Hladen
# Datum: 03.03.2025

# Ime log datoteke
$logFilePath = Join-Path -Path $PSScriptRoot -ChildPath "tnclog.txt"

# Zapisovanje v log datoteko
function Write-Log {
    param (
        [string]$Message
    )
    $Timestamp = Get-Date -Format "yyyy-MM-dd HH:mm:ss"
    $LogEntry = "[$Timestamp] $Message"
    Add-Content -Path $logFilePath -Value $LogEntry
    Write-Host $LogEntry
}

# Vnos IP naslovov, ločenih z vejico
$ipAddressesInput = Read-Host "Vnesi IP naslove, ki jih želiš preveriti. Če jih je
več, jih loči z vejico."
$ipAddresses = $ipAddressesInput.Split(',') | ForEach-Object { $_.Trim() }

# Vnos portov, ločenih z vejico
$portsInput = Read-Host "Vnesi porte, ki jih želiš preveriti. Če jih je več, jih
loči z vejico."
$ports = $portsInput.Split(',') | ForEach-Object { $_.Trim() }

# Zanka skozi IP naslove in porte
foreach ($ip in $ipAddresses) {
    foreach ($port in $ports) {
        try {
            # Izvedba TNC
            $testResult = Test-NetConnection -ComputerName $ip -Port $port -
InformationLevel Quiet
```

```
            # Zapisovanje rezultatov v log datoteko in izpis na zaslon
            if ($testResult) {
                Write-Log "IP: $ip, Port: $port - Odprt"
            } else {
                Write-Log "IP: $ip, Port: $port - Zaprto"
            }
        } catch {
            # Zapisovanje napak v log datoteko in izpis na zaslon
            Write-Log "Napaka pri preverjanju IP: $ip, Port: $port -
$($_.Exception.Message)"
        }
    }
}

Write-Log "Preverjanje portov končano."
```

Struktura skripte je enostavna: sestavljena je tako, da skripta najprej kreira log datoteko z imenom tnclog.txt v mapi, kjer se nahaja skripta. Sledi funkcija *Write-Log* za zapisovanje v log datoteko, nato pa skripta od uporabnika zahteva vnos IP-naslovov in portov. Pri obeh lahko uporabnik zapiše več vnosov, ločeni pa morajo biti z vejico. Sledita *foreach* zanki, v katerih se izvaja cmdlet *Test-NetConnection* na želenih IP-naslovih in portih. V primeru, da je do želenega IP-naslova omogočen dostop prek zapisanega porta, bo skripta izpisala, da je na določenem IP-naslovu določen port odprt, v nasprotnem primeru pa, da je zaprt. Če se v preverjanju pojavi napaka, se aktivira *catch* blok, kjer se izpiše, da je prišlo pri preverjanju do napake. Na koncu skripta še zapiše, da je preverjanje končano.

2.2.3 Python

Python je odprtokodni programski jezik, ki združuje različne stile programiranja – objektno orientiranega, proceduralnega in funkcijskega. (Fister, 2023, str. 41) Značilnost tega jezika je, da ne vsebuje deklaracij spremenljivk in da je za skriptno programiranje dokaj enostaven, vendar pa za naprednejšo uporabo vseeno potrebujemo določene osnove programskega jezika. Omogoča uporabo tako imenovanih modulov, ki v jeziku Python združujejo več programov, ki se nanašajo na določeno problemsko domeno. Prednost tega je v tem, da lahko ponovno uporabimo že napisano kodo, moduli pa prav tako omogočajo uporabo imenskih prostorov, kar pomeni, da se lahko imena funkcij znotraj različnih modulov tudi podvajajo. (Fister, 2023, str. 42–47)

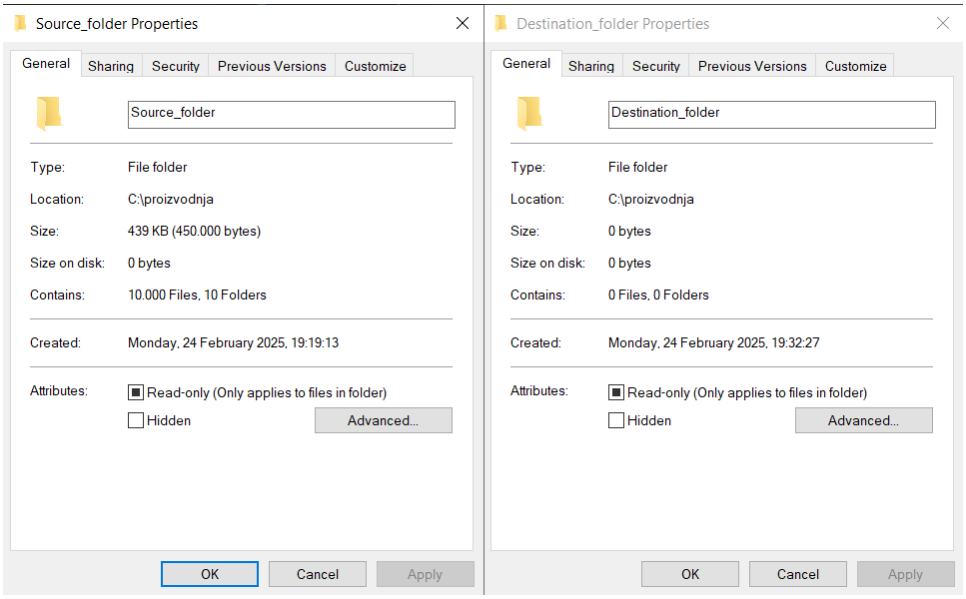
Dokaj preprost primer skripte v jeziku Python bi bil, ko v različnih »blokih« skripte določimo različne funkcije, glavna funkcionalnost skripte pa je varnostno kopiranje želene mape:

```
import os
import shutil
import datetime
```

```
import sys

# --- Konfiguracija ---

SOURCE_DIR = r"C:\proizvodnja\Source_folder"
BASE_DESTINATION_DIR = r"C:\proizvodnja\Destination_folder"
LOG_FILE = r"C:\proizvodnja\ARCHIVE\PRENOS_ARHIV\Logs\backup_log_dated.log"

# --- Funkcija za logging ---
def log_message(message, level="INFO"):
    try:
        # Preverjanje ali mapa za log datoteko obstaja
        log_dir = os.path.dirname(LOG_FILE)
        if log_dir and not os.path.exists(log_dir):
            try:
                os.makedirs(log_dir)
            except OSError as e:
                print(f"KRITIČNA NAPAKA: Ne morem ustvariti mape za log
'{log_dir}': {e}", file=sys.stderr)
                return

        # Dodajanje zapisa v datoteko
        with open(LOG_FILE, 'a', encoding='utf-8') as f:
            timestamp = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
            f.write(f"{timestamp} - {level} - {message}\n")
    except Exception as e:
        print(f"KRITIČNA NAPAKA: Ne morem pisati v log datoteko '{LOG_FILE}': {e}",
file=sys.stderr)

# --- Funkcija za varnostno kopiranje ---
def perform_backup(src, base_dst):

    # Pridobivanje trenutnega datuma v formatu YYYY-MM-DD
    today = datetime.date.today()
    date_str = today.strftime('%Y-%m-%d')

    archive_folder_name = f"ARHIV_ICT_{date_str}"

    # Ciljna mapa
    full_archive_path = os.path.join(base_dst, archive_folder_name)

    # Izvorna mapa
    source_folder_name = os.path.basename(src)
    if not source_folder_name:
        source_folder_name = "root_backup"

    final_destination = os.path.join(full_archive_path, source_folder_name)

    log_message(f"Začetek varnostnega kopiranja iz '{src}' v
'{final_destination}'.")

    # Preverjanje, ali izvorna mapa obstaja
    if not os.path.isdir(src):
        log_message(f"Izvorna mapa '{src}' ni najdena ali ni mapa. Varnostno
kopiranje prekinjeno.", "ERROR")
        return False

    try:
        os.makedirs(full_archive_path, exist_ok=True)
        log_message(f"Zagotovljen obstoj arhivske mape: '{full_archive_path}'",
"INFO")

        shutil.copytree(src, final_destination, dirs_exist_ok=True)
        log_message(f"Varnostno kopiranje uspešno zaključeno iz '{src}' v
'{final_destination}'.")
        return True

    # Predvidene osnovne napake
    except shutil.Error as e:
        log_message(f"Med operacijo kopiranja (shutil) je prišlo do napake: {e}",
"ERROR")
        return False
    except PermissionError as e:
        log_message(f"Dovoljenje zavrnjeno med operacijo kopiranja. Preverite
dovoljenja za '{src}' ali '{full_archive_path}'. Napaka: {e}", "ERROR")
        return False
    except OSError as e:
```

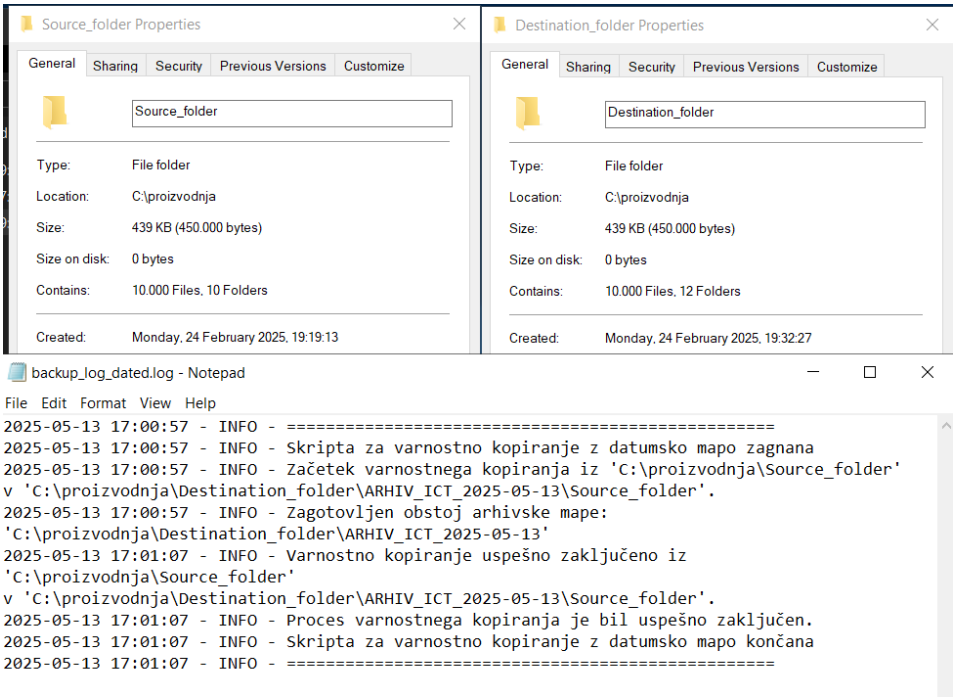
```
            log_message(f"Med operacijo kopiranja je prišlo do napake OS: {e}",
"ERROR")
            return False
        except Exception as e:
            log_message(f"Prišlo je do nepričakovane napake med kopiranjem: {e}",
"ERROR")
            return False

# --- Glavna funkcionalnost ---
if __name__ == "__main__":

    log_message("="*50, "INFO")
    log_message("Skripta za varnostno kopiranje z datumsko mapo zagnana", "INFO")

    # Izvajanje varnostnega kopiranja
    success = perform_backup(SOURCE_DIR, BASE_DESTINATION_DIR)

    # Zapisovanje končnega statusa
    if success:
        log_message("Proces varnostnega kopiranja je bil uspešno zaključen.",
"INFO")
    else:
        log_message("Proces varnostnega kopiranja je bil zaključen z napakami.
Preglejte zgornje zapise.", "WARNING")

    log_message("Skripta za varnostno kopiranje z datumsko mapo končana", "INFO")
    log_message("="*50, "INFO")
```

Skripta je zgrajena tako, da v prvem delu uvozimo potrebne module z uporabo import sistema. Ta poskrbi, da v svojo kodo uvozimo kodo, ki jo je nekdo že napisal. V našem primeru smo uvozili module os, shutil, datetime in sys, ki nam omogočajo različne funkcionalnosti, od interakcije z operacijskim sistemom do dela s časom in datumom. V naslednjem koraku smo definirali spremenljivke poti, ki jih bo skripta uporabljala – izvorno mapo, ciljno mapo in mapo za dnevniško datoteko. Sledi funkcija za beleženje v dnevniško datoteko. Ta funkcija preveri, ali log datoteka v specificirani mapi obstaja, in jo ustvari, če še ne obstaja. Nato jo odpre in na konec datoteke doda nove vrstice z informacijami o kopiranju datotek in morebitnih napakah. Temu sledi funkcija varnostnega kopiranja, ki najprej pripravi ciljne mape, zapiše v dnevniško datoteko, da se je kopiranje začelo, preveri, ali izvorna mapa obstaja, nato pa začne kopirati vsebino izvorne mape v ciljno mapo. Ob uspešnem kopiranju bo skripta to sporočilo zapisala v dnevniško datoteko. Če bo med postopkom kopiranja prišlo do specifičnih napak, na primer težave z dovoljenji ali drugimi sistemskimi napakami, bo skripta te napake ujela in zapisala podrobnosti v dnevniško datoteko, funkcijo pa končala z neuspehom. Avtomatizacijo te skripte lahko uredimo z uporabo Task Schedulerja v okolju Windows ali pa z uporabo Cron v okolju Linux. V primeru okolja Linux bi seveda morali popraviti poti izvorne in ciljne mape ter mape dnevniške datoteke.



Slika 6: Stanje v izvorni in ciljni mapi pred zagonom skripte

Vir: Lasten



Slika 7: Stanje v izvorni in ciljni mapi po zagonu skripte in vsebina dnevniške datoteke

Vir: Lasten

2.2.4 Ansible

Ansible je odprtokodno orodje, napisano v jeziku Python, ki ga uporabljamo za avtomatizacijo IT-nalog. Glavni namen je avtomatizacija ponavljajočih se nalog, kar zmanjšuje možnost

človeških napak. Ansible se od konkurenčnih orodij pogosto razlikuje po tem, da ne potrebuje agentov – to pomeni, da nam na ciljnih sistemih ni treba nameščati dodatne programske opreme. (Introduction to Ansible, 2025) To v praksi pomeni, da potrebujemo le en strežnik, na katerega namestimo orodje Ansible, nato pa prek tega strežnika in povezave SSH upravljamo vse ostale, ki nam to omogočajo.

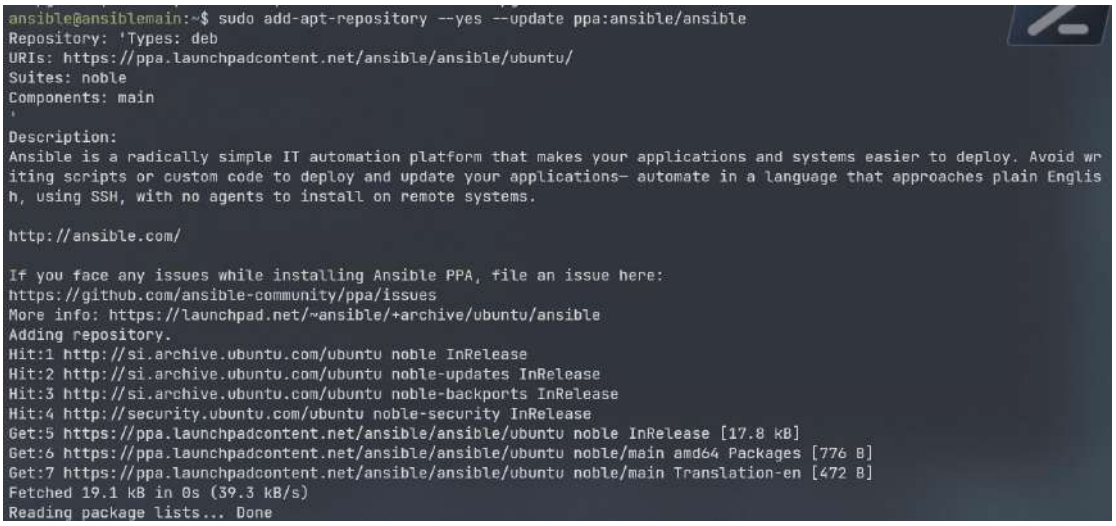
Ansible za svoje delovanje uporablja tako imenovane *Playbooke*, ki so zapisani v obliki YAML, kar je primerno za začetnike, saj je prag vstopa dokaj nizek. Ključne komponente *Playbooka* so:

1. Hosts – ta komponenta so ciljni sistemi za izvedbo nalog, lahko so posamezni sistemi, lahko pa so skupine sistemov;
2. Tasks – ta komponenta vsebuje seznam nalog, ki jih bo orodje Ansible izvajalo na ciljnih sistemih;
3. Vars – ta komponenta definira spremenljivke, ki jih je mogoče uporabiti v nalogah;
4. Handlers – v tej komponenti so posebne naloge, ki se izvajajo le v primeru, da jih sproži druga naloga.
5. Roles – ta komponenta nam omogoča organizacije Playbooka v logične enote.

Kljub relativni enostavni uporabi, aktivni skupnosti in nizkemu pragu vstopa pa ima orodje Ansible tudi nekatere slabosti. Kompleksni scenariji zahtevajo zapletene Playbooke, odvisnost od povezave SSH pa lahko v podjetjih s strogimi varnostnimi politikami predstavlja določeno mero tveganja.

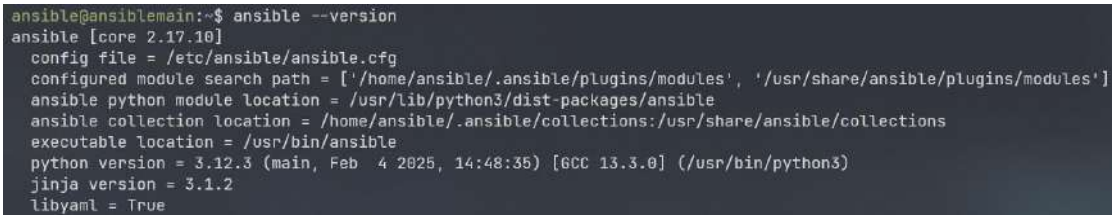
Namestitev orodja je preprosta: v ta namen smo pripravili štiri strežnike, na katere je nameščen operacijski sistem Ubuntu Server 24.04 LTS. Ker orodje za svoje delovanje ne zahteva ogromno računalniških virov, smo za strežnike uporabili po 2 jedri CPU, 2GB pomnilnika in 20GB trdega diska.

Na glavnem strežniku je bilo najprej treba dodati repozitorij Ansible, kar smo naredili z uporabo ukaza »sudo add-apt-repository --yes --update ppa:ansible/ansible«, nadaljevali pa smo z ukazom »sudo apt install ansible«. Uspešno namestitev smo na koncu preverili z uporabo ukaza »ansible –version«.



Slika 8: Dodajanje repozitorija Ansible

Vir: Lasten



Slika 9: Različica Ansible

Vir: Lasten

V hosts datoteki smo nato dodali naše strežnike, prav tako pa uporabniško ime in geslo za uporabnika, ki smo ga kreirali za namene uporabe orodja Ansible.

Slika 10: Konfiguracija Ansible hosts

Vir: Lasten

Prvi ukaz za preverjanje delovanja sistema je uporaba modula »ping«, za kar smo uporabili ukaz »ansible linux_servers -m ping«. S tem smo definirali, da naj Ansible izvede modul ping

na skupino »linux_servers«, ki je definirana v datoteki hosts. Rezultat tega ukaza je prikazan na naslednji sliki.

```
ansible@ansiblemain:/etc/ansible$ ansible linux_servers -m ping
192.168.233.132 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3.12"
    },
    "changed": false,
    "ping": "pong"
}
192.168.233.131 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3.12"
    },
    "changed": false,
    "ping": "pong"
}
192.168.233.133 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3.12"
    },
    "changed": false,
    "ping": "pong"
}
```

Slika 11: Ansible ping

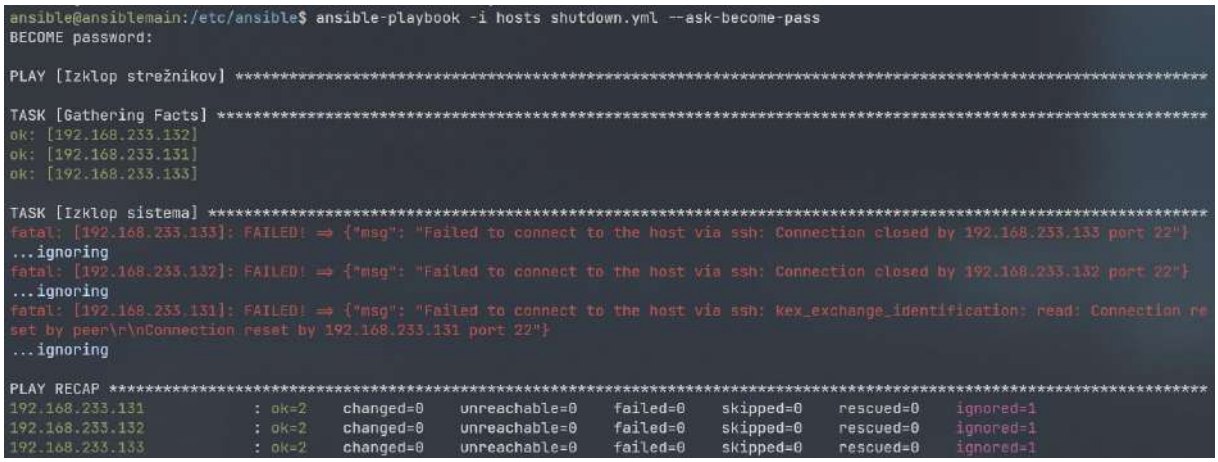
Vir: Lasten

Kreirali smo tudi testni Playbook za izklop vseh strežnikov naenkrat. Kot vidimo, smo mu definirali ime, ciljne strežnike, da lahko ukaz zažene kot skrbnik sistema, nato pa definirali še samo nalogo. Ta zažene ukaz »sudo shutdown -h now«, nakar nas sistem vpraša za administratorsko geslo, ob vnosu gesla pa izpiše rezultat izvedbe Playbooka.

```
---
- name: Izklop strežnikov
  hosts: linux_servers
  become: yes
  tasks:
    - name: Izklop sistema
      command: shutdown -h now
      ignore_errors: yes
```

Slika 12: Izklop sistema Playbook

Vir: Lasten



Slika 13: Rezultat Playbooka

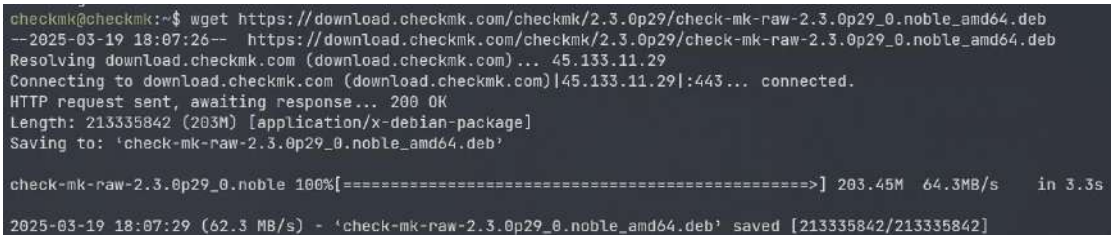
Vir: Lasten

2.2.5 Checkmk

Checkmk je orodje za nadzor IT-infrastrukture, ki nam omogoča spremljanje delovanja naprav v naši infrastrukturi. To orodje sistemskim administratorjem ponuja celovit nadzor nad infrastrukturo, saj z zbiranjem in analiziranjem podatkov o stanju in zmogljivosti naprav omogoča hitro zaznavanje težav, posledično pa avtomatizira proces obveščanja sistemskih administratorjev o stanju naprav. (Checkmk 2.3, 2025)

Postopek namestitve tega orodja je precej preprost. Najprej potrebujemo strežnik, na katerem bo storitev Checkmk zagnana – uporabimo lahko strežnik Linux Ubuntu ali pa storitev namestimo z Dockerjem.

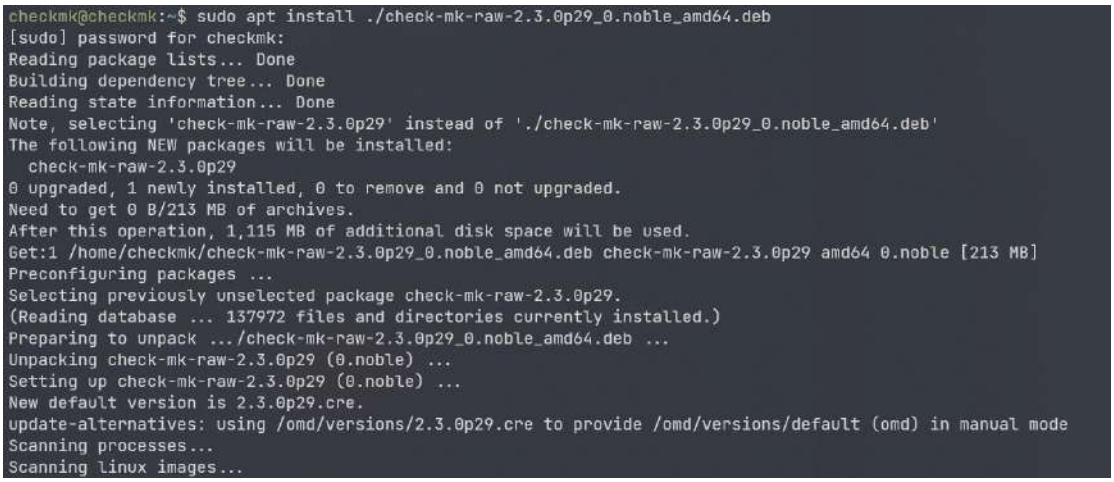
Za testno namestitev smo uporabili Ubuntu Server 24.04 LTS, ki smo mu dodelili 2 jedri CPU, 2 GB pomnilnika in disk v velikosti 20 GB.



Slika 14: Pridobivanje orodja Checkmk

Vir: Lasten

Iz zgornje slike je razviden enostaven proces pridobivanja potrebnih instalacijskih datotek za namestitev orodja Checkmk. Za to smo uporabili ukaz *wget*.

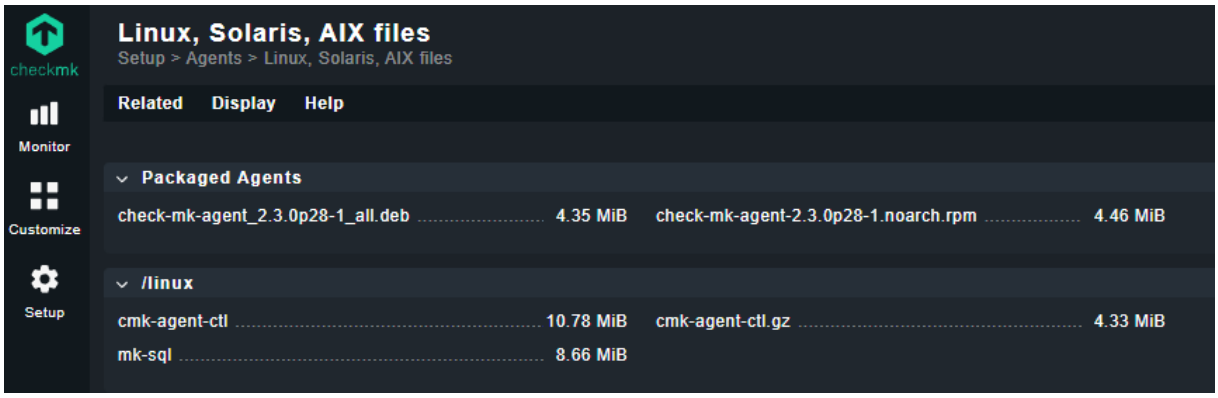


Slika 15: Namestitev orodja Checkmk

Vir: Lasten

Za namestitev uporabimo ukaz *sudo apt install ./check…,* saj za namestitev potrebujemo administratorske pravice. Temu sledi kreiranje tako imenovanega nadzornega mesta (monitoring site) z ukazom *sudo omd create $monitoring_site_name*, nato pa sledi *sudo omd start $monitoring_site_name*. Vmes nam sistem izpiše lokacijo, kje v našem omrežju se to mesto nahaja, prav tako dobimo geslo za vpis v spletno mesto.

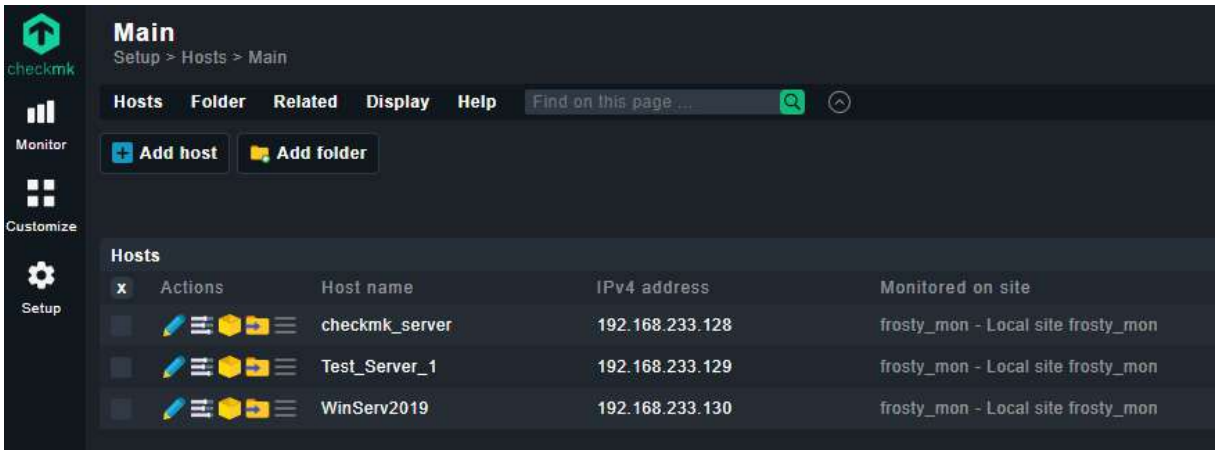
Ko se prijavimo v spletno mesto, bo to izgledalo dokaj prazno, dokler ne dodamo nekaj »klientov«, ki jih Checkmk spremlja. V vmesniku imamo navodila za dostop do tako imenovanih agentov, ki jih namestimo na kliente.



Slika 16: Prikaz pridobivanja agentov v orodju Checkmk

Vir: Lasten

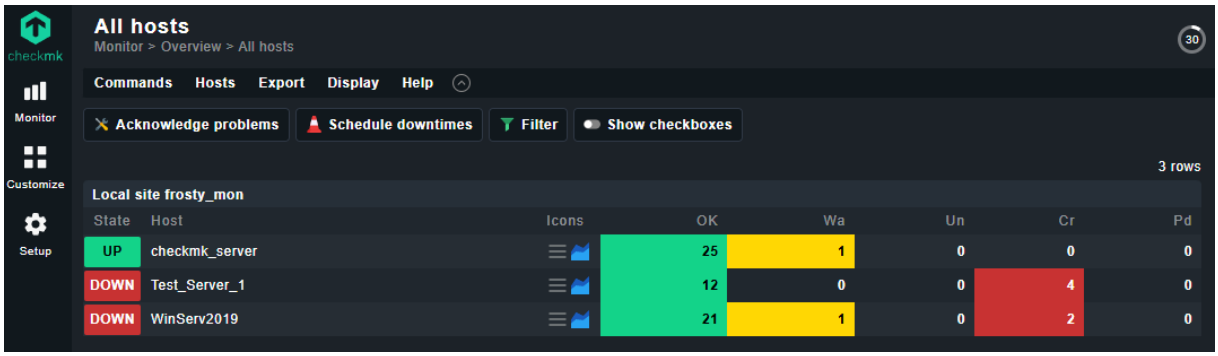
Ko na klientu namestimo agenta, ga lahko dodamo v glavno Checkmk aplikacijo.



Slika 17: Prikaz dodanih klientov v orodju Checkmk

Vir: Lasten

Ko imamo v aplikacijo dodane kliente, jih lahko spremljamo prek več različnih prikazov. En od njih nam prikazuje stanje klientov, prav tako nam prikaže morebitna opozorila in kritične napake.



Slika 18: Prikaz stanja klientov v orodju Checkmk

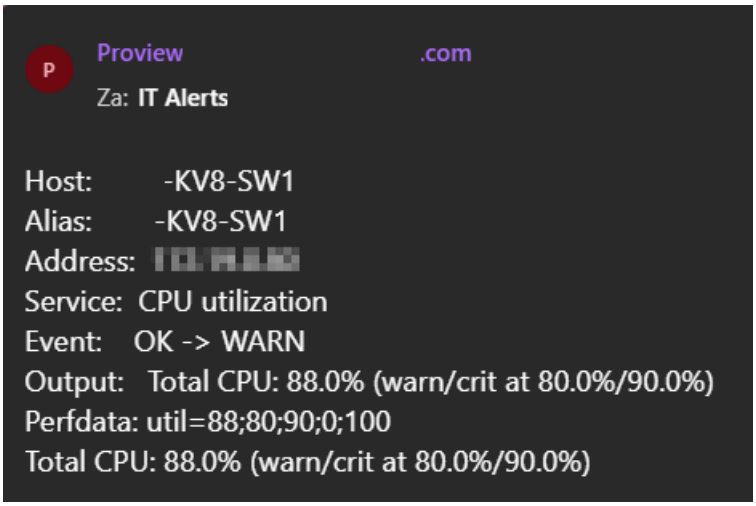
Vir: Lasten

Orodje nam omogoča tudi kreiranje lastnih pravil, na podlagi katerih nas avtomatsko opozori v primeru napak ali težav.

Slika 19: Pravilo za obveščanje v orodju Checkmk

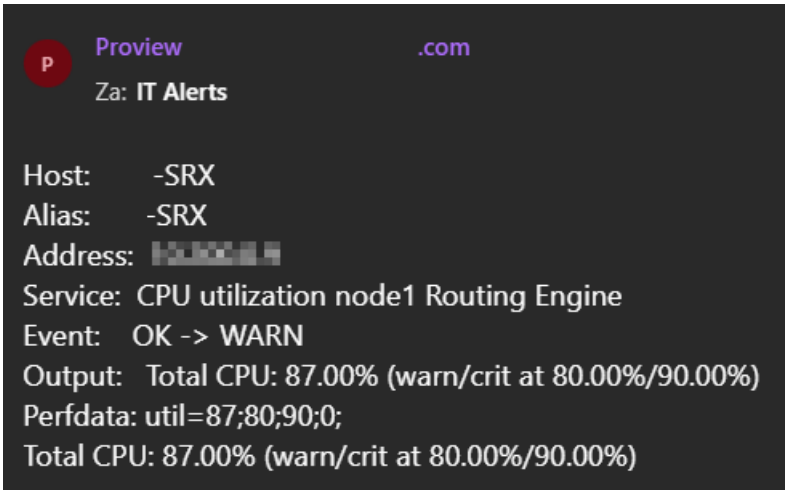
Vir: Lasten

Na zgornji sliki je vidno pravilo, ki bo sistemskega administratorja, če je na strežniku omogočeno pošiljanje e-pošte, opozorilo v primeru, da CPU load na klientu Test_Server_1 ali checkmk_server doseže kritično raven.



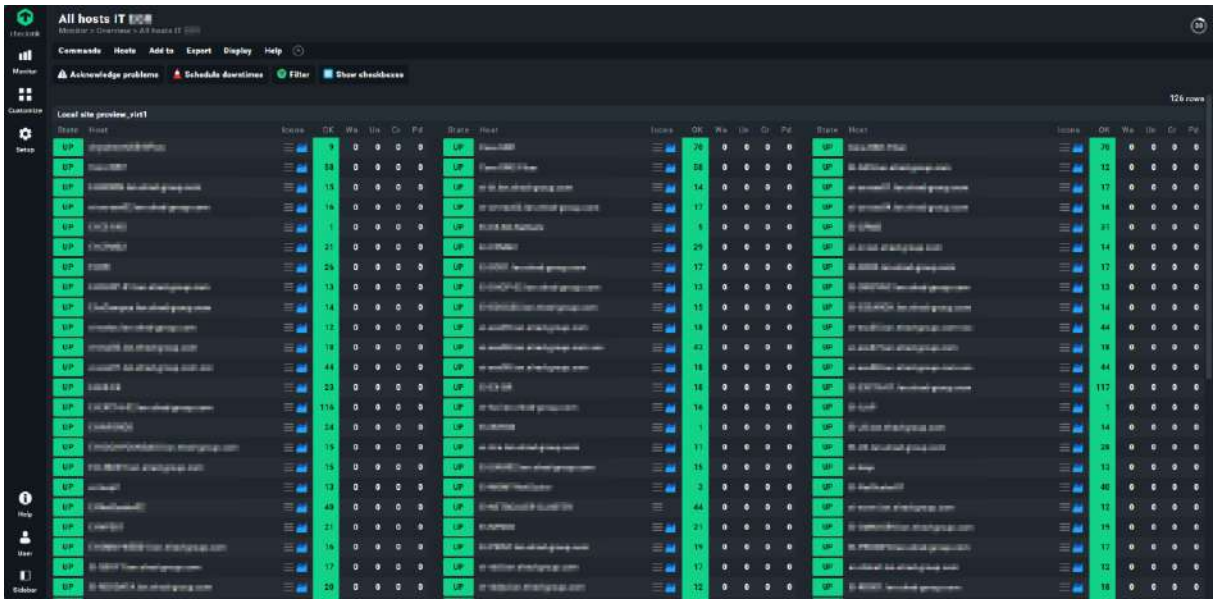
Slika 20: Primer opozorila orodja Checkmk prek e-pošte 1

Vir: Lasten



Slika 21: Primer opozorila orodja Checkmk prek e-pošte 2

Vir: Lasten



Slika 22: Primer prikaza stanja klientov v orodju Checkmk v podjetju

Vir: Lasten

Iz zgornjih slik je razvidno, da lahko s pravilno konfiguracijo orodja Checkmk in tudi konfiguracijo pošiljanja e-poštnih sporočil sistemski administratorji hitreje odreagirajo ob morebitnih težavah v IT-infrastrukturi podjetja.

2.3 Umetna inteligenca v sistemski administraciji

Umetna inteligenca prinaša velike spremembe na področje sistemske administracije, saj prinaša nove možnosti avtomatizacije, optimizacije in izboljšanja zanesljivosti informacijskih sistemov. Če želimo prepoznati potencial umetne inteligence in to učinkovito implementirati v praksi, pa je potrebno razumevanje določenih temeljnih konceptov.

Umetna inteligenca je dokaj široko področje, ki se ukvarja z razvojem računalniških sistemov in strojev, ki so sposobni izvajanja nalog, ki običajno zahtevajo človeški dejavnik. Med te naloge spadajo sklepanje, učenje, razumevanje jezika in reševanje problemov. V poslovnem okolju med orodja umetne inteligence spadajo orodja, ki računalnikom omogočajo izvajanje naprednih funkcij, vključno z analizo podatkov, razumevanjem govornega in pisnega jezika in podobno. Čeprav se podrobnosti med različnimi metodami umetne inteligence razlikujejo, pa je osnova v večini enaka – velike količine podatkov. Sistemi umetne inteligence se na podlagi ogromne količine podatkov učijo, prepoznavajo razne vzorce in povezave med podatki, ki jih človeško oko morda prezre. (What is Artificial Intelligence (AI)?, 2025)

Ko govorimo o umetni inteligenci, ne smemo pozabiti ključne veje umetne inteligence, to je strojnega učenja in podvrste le-tega, to je globokega učenja. Pri tej veji umetne inteligence sistemi niso eksplicitno programirani za vsako nalogo, temveč se učijo iz podatkov. Algoritmi analizirajo velike količine podatkov, prepoznajo vzorce in na podlagi teh vzorcev sprejemajo odločitve. (Deep Learning in Cybersecurity: Threat Detection and Defense, 2025) V kontekstu sistemske administracije to pomeni, da lahko sistemi strojnega učenja napovedo morebitne okvare strojne opreme na podlagi analize preteklih podatkov o delovanju, prav tako se taki sistemi uporabljajo pri rešitvah, kot sta Darktrace in QRadar.

Ker umetna inteligenca, kot že omenjeno, omogoča samodejno odkrivanje anomalij, omogoča ta zmožnost IT-ekipam možnost ukrepanja, preden majhne težave preidejo v večje težave in izpade. AIOps izkorišča strojno učenje za obdelavo podatkov za odkrivanje anomalij v realnem času, zgodnje in natančno odkrivanje le-teh pa je prvi korak k učinkovitemu reševanju težav. (What is AIOps?, 2025)

2.4 Avtomatizacija in varnostna skladnost

Ob današnjih vedno številčnejših sistemih in povezavah med njimi je bistvenega pomena smiselno in varno urediti avtomatizacijo upravljanja uporabniških računov in pristopov ter s tem zagotoviti informacijsko varnostno skladnost podjetja.

Ročno upravljanje uporabniških računov prinaša številna tveganja, ki lahko resno ogrozijo varnost in skladnost organizacije. Med najpogostejše napake spadajo tipkarske napake pri vnosu podatkov, napačne dodelitve pravic za dostope do podatkov in pozabljeni odvzemi pravic uporabnikom, ki so zapustili podjetje. (How Automation Simplifies User Access Reviews for Remote and Hybrid Workforces, 2025) Z avtomatizacijo upravljanja uporabniških računov pa lahko tako že vnaprej določimo točno določene pravice, ki jih nov uporabnik na podlagi delovnega mesta in delovnih nalog potrebuje – te spreminjamo ob morebitnem napredovanju uporabnika, prav tako pa ob integraciji z drugimi sistemi avtomatiziramo deaktivacijo uporabniškega računa v primeru, da uporabnik zapusti podjetje.

3 AVTOMATIZACIJA SISTEMSKIH OPRAVIL

3.1 Primeri avtomatizacije s PowerShellom

3.1.1 Brisanje specifičnega direktorija s pomočjo PowerShell skripte in Task Schedulerja

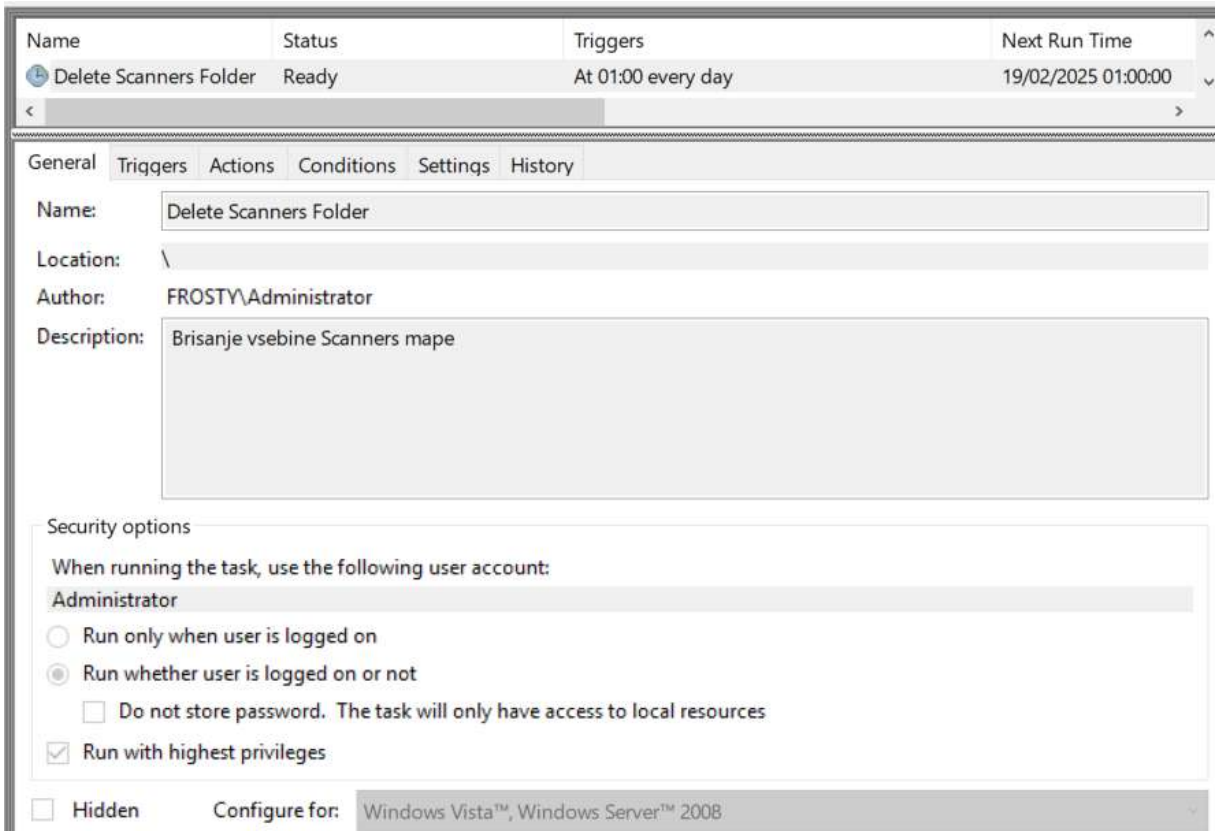
```
#
# PowerShell skripta za brisanje datotek iz izvorne mape in podmap
#
# Datum: 18.02.2025
# Avtor: Iztok Hladen
# Za testiranje napak odkomentiraj Write-Verbose in Write-Host vrstice

# Pot do mape Scanners (spremeni, če je potrebno)
$sourceFolder = "\\10.0.1.158\Scanners"

# Pot do log datoteke
$logFilePath = "C:\Logs\delete_scanners_folder.log"

# Preverjanje ali mapa Scanners obstaja
if (!(Test-Path -Path $sourceFolder -PathType Container)) {
    $errorMessage = "Mapa '$sourceFolder' ne obstaja!"
    Write-Error $errorMessage
    Add-Content -Path $logFilePath -Value "$(Get-Date -Format 'yyyy-MM-dd
HH:mm:ss') - ERROR: $errorMessage"
    exit
}

# Zapisovanje v log datoteko
Add-Content -Path $logFilePath -Value "$(Get-Date -Format 'yyyy-MM-dd HH:mm:ss') -
Skripta za brisanje datotek se je začela."

# Števec napak
$errorsOccurred = $false

# Seznam vseh datotek v mapi in podmapah
$filesToDelete = Get-ChildItem -Path $sourceFolder -Recurse -File

# Brisanje datotek
foreach ($file in $filesToDelete) {
    try {
        Remove-Item -Path $file.FullName -Force -ErrorAction Stop
        # Write-Verbose "Izbrisana datoteka: $($file.FullName)"
    }
    catch {
        $errorsOccurred = $true # Beleženje napake
        $warningMessage = "Napaka pri brisanju datoteke: $($file.FullName) -
$($_.Exception.Message)"
        Write-Warning $warningMessage
        Add-Content -Path $logFilePath -Value "$(Get-Date -Format 'yyyy-MM-dd
HH:mm:ss') - OPOZORILO: $warningMessage"
    }
}

if ($errorsOccurred) {
    Add-Content -Path $logFilePath -Value "$(Get-Date -Format 'yyyy-MM-dd
HH:mm:ss') - Skripta za brisanje datotek se je končala z OPOZORILI."
} else {
    Add-Content -Path $logFilePath -Value "$(Get-Date -Format 'yyyy-MM-dd
HH:mm:ss') - Skripta za brisanje datotek se je uspešno končala."
}

# Write-Host "Skripta za brisanje datotek v mapi '$sourceFolder' je končana."
# Write-Host "Struktura map je ostala nespremenjena."
```

Skripta je enostaven primer brisanja vsebine specifične mape. V trenutni izvedbi jo uporabljamo v namene brisanja že skeniranih dokumentov, ki jih uporabniki po skeniranju shranijo v svojo lokalno shrambo oziroma na strežnik. Namen skripte je tako brisanje dokumentov, ki niso več v uporabi, s tem pa povečamo preglednost v ciljni mapi in prav tako privarčujemo s prostorom na strežniku.

Skripta je sestavljena iz nekaj enostavnih sklopov:

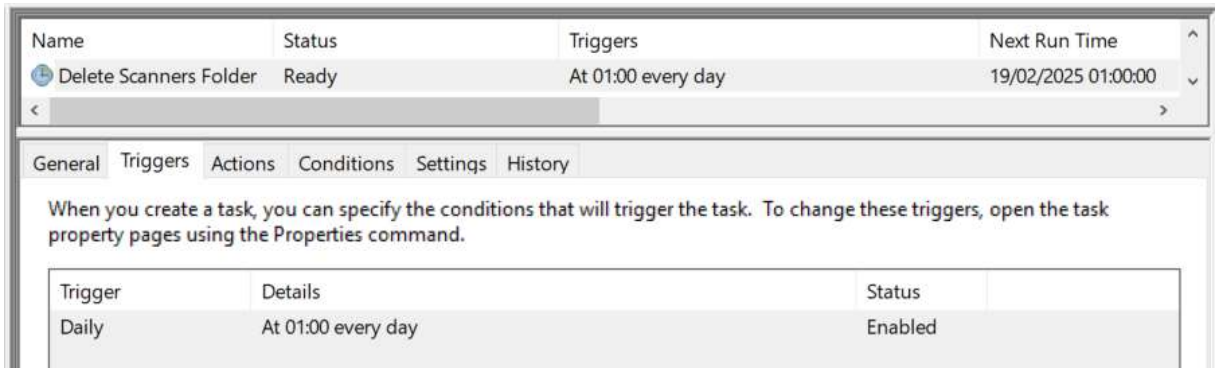
1. v prvem sklopu definiramo pot do mape, iz katere želimo brisati datoteke, in pot do mape, v katero bomo zapisovali log datoteko;
2. v drugem sklopu preverjamo, ali izvorna mapa (v našem primeru \\10.0.1.158\Scanners) sploh obstaja. Tu se pojavi prvo preverjanje, ali je skripta pravilno zastavljena. V primeru, da mapa ne obstaja (preverjanje z »if« stavkom in negacijo »!«), nam skripta zapiše v log datoteko sporočilo napake;
3. sledi začetek zapisovanja v log datoteko, v katero se najprej zapiše, kdaj se je skripta začela izvajati v formatu »leto-mesec-dan in ura-minuta-sekunda«;
4. v četrtem sklopu se definira števec napak na vrednost *$false*. Če bo v prihodnosti prišlo do kakršnekoli napake, se bo ta vrednost spremenila na *$true*;
5. v petem sklopu skripta pridobi seznam vseh datotek (z uporabo Get-ChildItem in -File) v glavni mapi in podmapah (z uporabo -Recurse), ki jih je treba izbrisati;
6. v šestem sklopu se zažene zanka *foreach* za brisanje vsake datoteke. Blok *try catch* je v našem primeru prisoten za obravnavo napak. Če se v *try* bloku pojavi napaka, se bo v *catch* bloku spremenljivka *$errorsOccured* nastavila na *true*, hkrati pa se bo ustvarilo sporočilo o napaki v log datoteki;
7. v zadnjem, sedmem sklopu se zaključi zapisovanje v log datoteko s sporočilom o koncu izvajanja skripte.

Sledi še prikaz nastavitev Task Schedulerja.



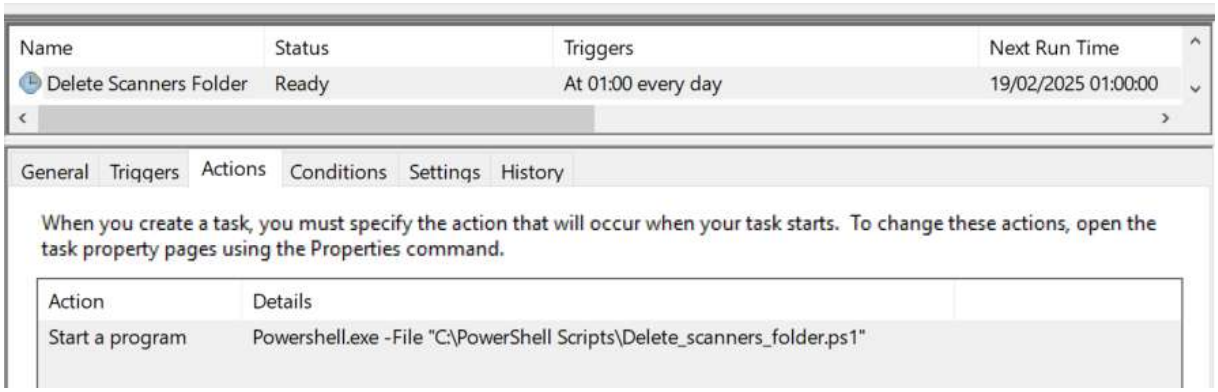
Slika 23: Splošne nastavitve Task Schedulerja – PowerShell brisanje vsebine mape

Vir: Lasten



Slika 24: Trigger nastavitve Task Schedulerja – PowerShell brisanje vsebine mape

Vir: Lasten



Slika 25: Actions nastavitve Task Schedulerja – PowerShell brisanje vsebine mape

Vir: Lasten

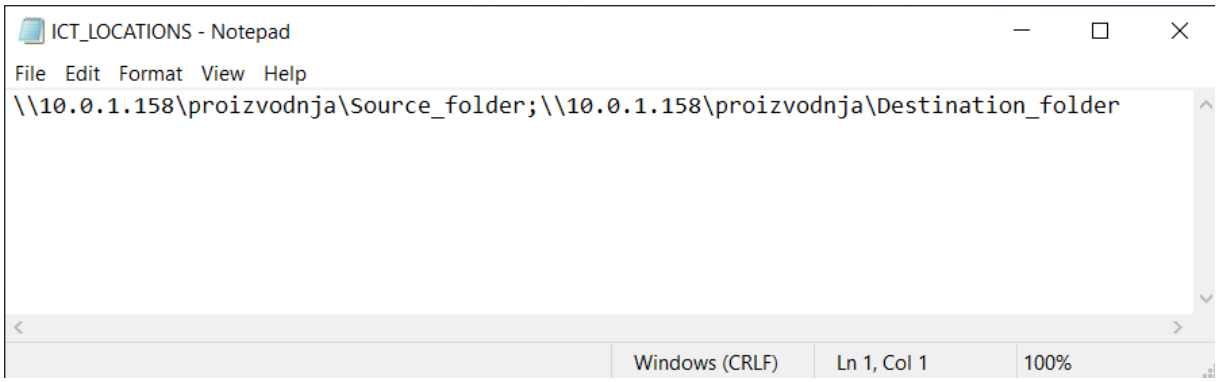
3.1.2 Premikanje datotek v arhivsko mapo s pomočjo PowerShella

```
#
# Premikanje datotek z uporabo orodja Robocopy
#
# Avtor: Iztok Hladen
# Datum: 20.01.2025
#
# Določimo poti do datotek
$source_dest_file =
"\\10.0.1.158\proizvodnja\ARCHIVE\PRENOS_ARHIV\ICT_LOCATIONS.txt"
$date = Get-Date -Format "yyyy-MM-dd"
$log_file = "\\10.0.1.158\proizvodnja\ARCHIVE\PRENOS_ARHIV\Logs\ICT_LOG_$date.txt"

# Preberemo poti iz datoteke
$paths = Get-Content -Path $source_dest_file

foreach ($path in $paths) {
    $split_path = $path -split ";"
    $source_folder = $split_path[0]
    $destination_folder = $split_path[1]

    # Preverimo, ali oba folderja obstajata
    $source_exists = Test-Path -Path $source_folder
    $destination_exists = Test-Path -Path $destination_folder

    if (-not $source_exists -or -not $destination_exists) {
        # Zapišemo napako v log datoteko
        $error_message = "Napaka: "
        if (-not $source_exists) {
            $error_message += "Source folder ne obstaja: $source_folder. "
        }
        if (-not $destination_exists) {
            $error_message += "Destination folder ne obstaja: $destination_folder."
        }
        Add-Content -Path $log_file -Value $error_message
        continue
    }

    # Preverimo, ali sta source in destination folder enaka
    if ($source_folder -eq $destination_folder) {
        # Zapišemo napako v log datoteko
        $error_message = "Napaka: Source in destination folder sta enaka:
$source_folder."
        Add-Content -Path $log_file -Value $error_message
        continue
    }

    # Zapišemo datum začetka v log datoteko
    $start_date = Get-Date
```

```
    Add-Content -Path $log_file -Value "Datum začetka: $start_date"

    # Zapišemo ime source folderja v log datoteko
    Add-Content -Path $log_file -Value "Source folder: $source_folder"

    # Preštejemo .txt datoteke v source folderju in zapišemo število v log datoteko
    $file_count_before = (Get-ChildItem -Path $source_folder -Recurse -File -Filter
"*.txt").Count
    Add-Content -Path $log_file -Value "Število .txt datotek pred premikanjem:
$file_count_before"

    # Ustvarimo novo mapo v destination folderju z imenom "ARHIV_ICT_$date"
    $archive_folder = "$destination_folder\ARHIV_ICT_$date"
    New-Item -ItemType Directory -Path $archive_folder

    # Ustvarimo enako strukturo podmap v destination folderju
    Get-ChildItem -Path $source_folder -Directory | ForEach-Object {
        $subfolder = $_.FullName.Substring($source_folder.Length)
        New-Item -ItemType Directory -Path "$archive_folder$subfolder"
    }

    # Premaknemo samo .txt datoteke z uporabo robocopy
    $robocopy_log =
"\\10.0.1.158\proizvodnja\ARCHIVE\PRENOS_ARHIV\Logs\ICT_robocopy_log_$date.txt"
    robocopy $source_folder $archive_folder *.txt /E /MOV /MT:8 /R:10 /W:20
/log+:$robocopy_log /nfl /np

    # Preštejemo .txt datoteke v destination folderju po premikanju
    $file_count_after = (Get-ChildItem -Path $archive_folder -Recurse -File -Filter
"*.txt").Count
    Add-Content -Path $log_file -Value "Destination folder: $archive_folder"
    Add-Content -Path $log_file -Value "Število prekopiranih .txt datotek:
$file_count_after"

    # Zapišemo čas konca v log datoteko
    $end_date = Get-Date
    Add-Content -Path $log_file -Value "Čas konca: $end_date"

    # Preverimo napake in jih zapišemo v log datoteko
    $errors = Select-String -Path $robocopy_log -Pattern "ERROR"
    if ($errors) {
        Add-Content -Path $log_file -Value "Napake: $errors"
    }

    # Dodamo prazno vrstico na koncu log datoteke
    Add-Content -Path $log_file -Value ""
}
```

Skripta je aktualen primer avtomatizacije prenosa podatkov v živem okolju. Razdeljena je na več logičnih delov, uporablja pa orodje Robocopy za premikanje datotek in hkrati ohranja enako strukturo podmap tako v izvorni kot v ciljni mapi. Skripta je uporabna v primerih, ko določenih informacij ne moremo izvoziti v bazo SQL, vendar moramo datoteke vseeno hraniti določeno število let.

V skripti najprej definiramo datoteko, iz katere skripta bere pot izvorne in pot ciljne mape, nato pa definiramo pot dnevniške datoteke. Prav tako za potrebe avtomatiziranega poimenovanja definiramo še trenutni datum z uporabo ukaza *Get-Date*. Skripta nato bere poti iz datoteke, poimenovane ICT_LOCATIONS.txt, v kateri sta poti ločeni s podpičjem.

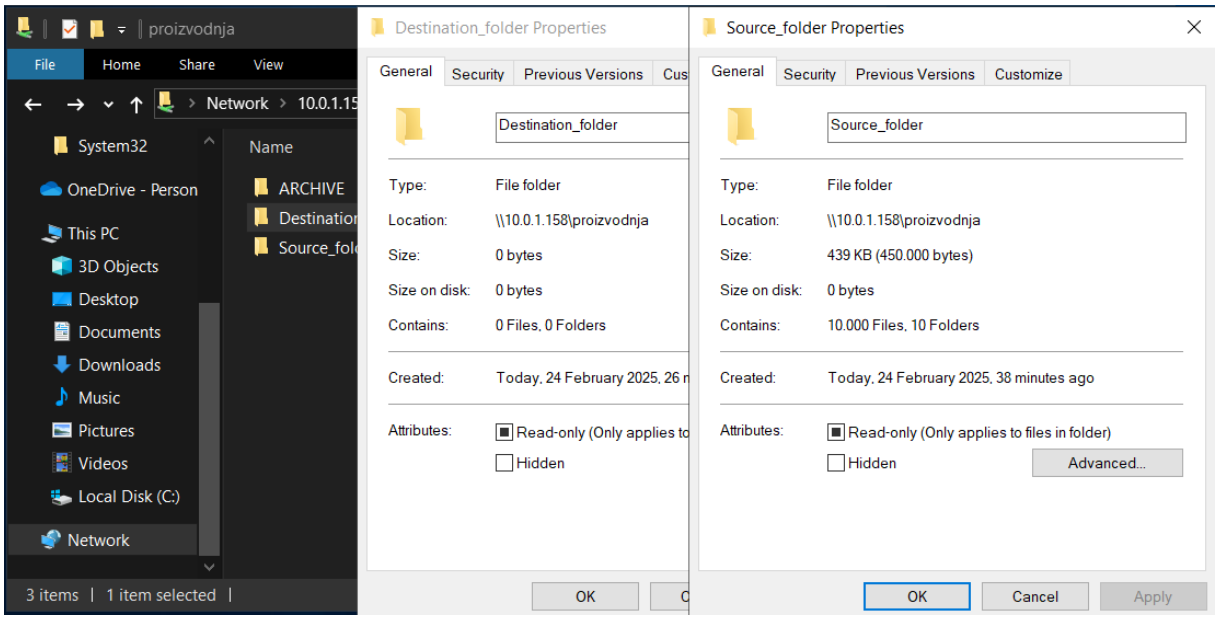


Slika 26: Primer vsebine datoteke ICT_LOCATIONS.txt

Vir: Lasten

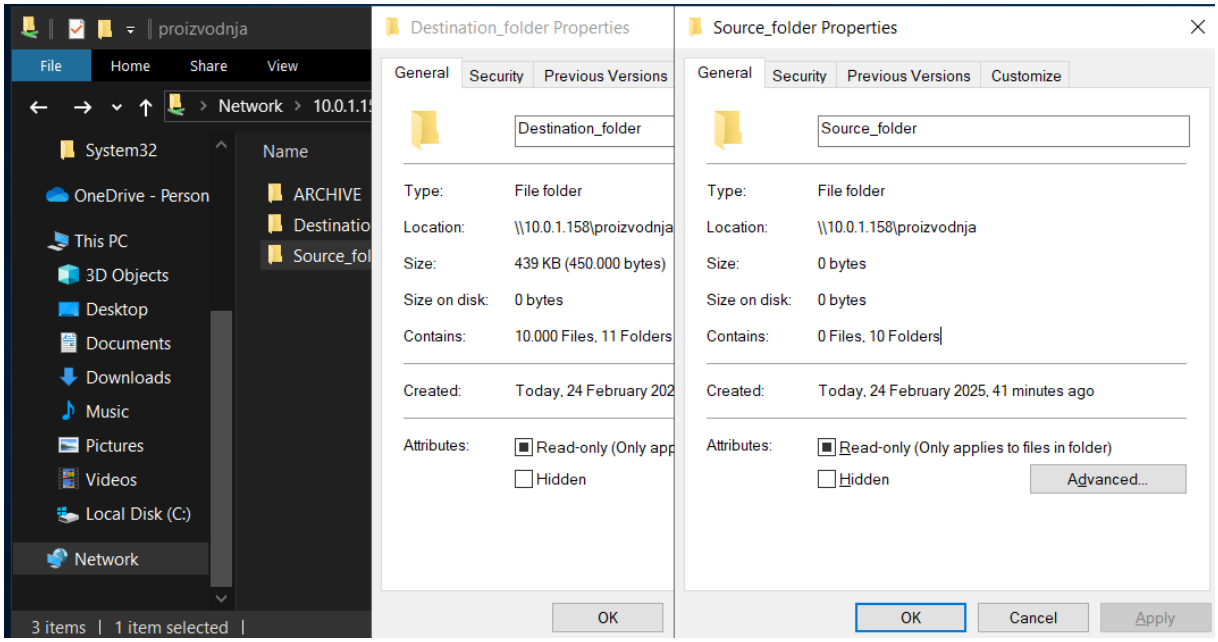
Sledi *foreach* zanka, v kateri skripta procesira en par izvorne in ciljne mape. Poti nato razdeli na dva dela z uporabo ukaza *-split*, ločilo pa je definirano kot podpičje. Vrednosti nato shrani v spremenljivki *$source_folder* in *$destination_folder*. Sledi kratko preverjanje, ali izvorna in ciljna mapa sploh obstajata ter ali sta morda po uporabniški napaki vneseni isti lokaciji za izvorno in ciljno mapo. Sledita štetje datotek v izvorni mapi in njenih podmapah ter beleženje v dnevniško datoteko, nato pa je na vrsti ustvarjanje arhivske mape v ciljni mapi, v kateri ohrani strukturo podmap iz izvorne mape. Nato z uporabo ukaza *robocopy* začne premikati datoteke iz izvorne v ciljno mapo. Uporablja vrsto parametrov, s katerimi smo določili premikanje map (/MOV), uporabo več niti za hitrejše premikanje (/MT:8), v primeru napake pa ponovni poskus do desetkrat (/R:10), z razmikom 20 sekund med poskusi (/W:20). Kreira tudi svojo dnevniško datoteko, saj omogoča preglednejši zapis morebitnih napak. Skripta nato prešteje še število datotek v ciljni mapi in to tudi zapiše v dnevniško datoteko. S tem lahko sistemski administrator hitro preveri, ali je morda prišlo do napak pri prenosu. Skripto po želji s pomočjo Task Schedulerja lahko izvajamo avtomatsko ob določenem dnevu in uri – ker gre za izvajanje v živem okolju je to lahko na primer ob nedeljah ob 8. uri zjutraj, ko proizvodnja miruje.

Vsekakor je za velike količine datotek smiselno uporabiti določeno vrsto avtomatizacije, saj s tem bistveno prihranimo na času, ki bi ga po nepotrebnem porabili, če bi namesto skripte ročno premikali posamezne datoteke in mape.



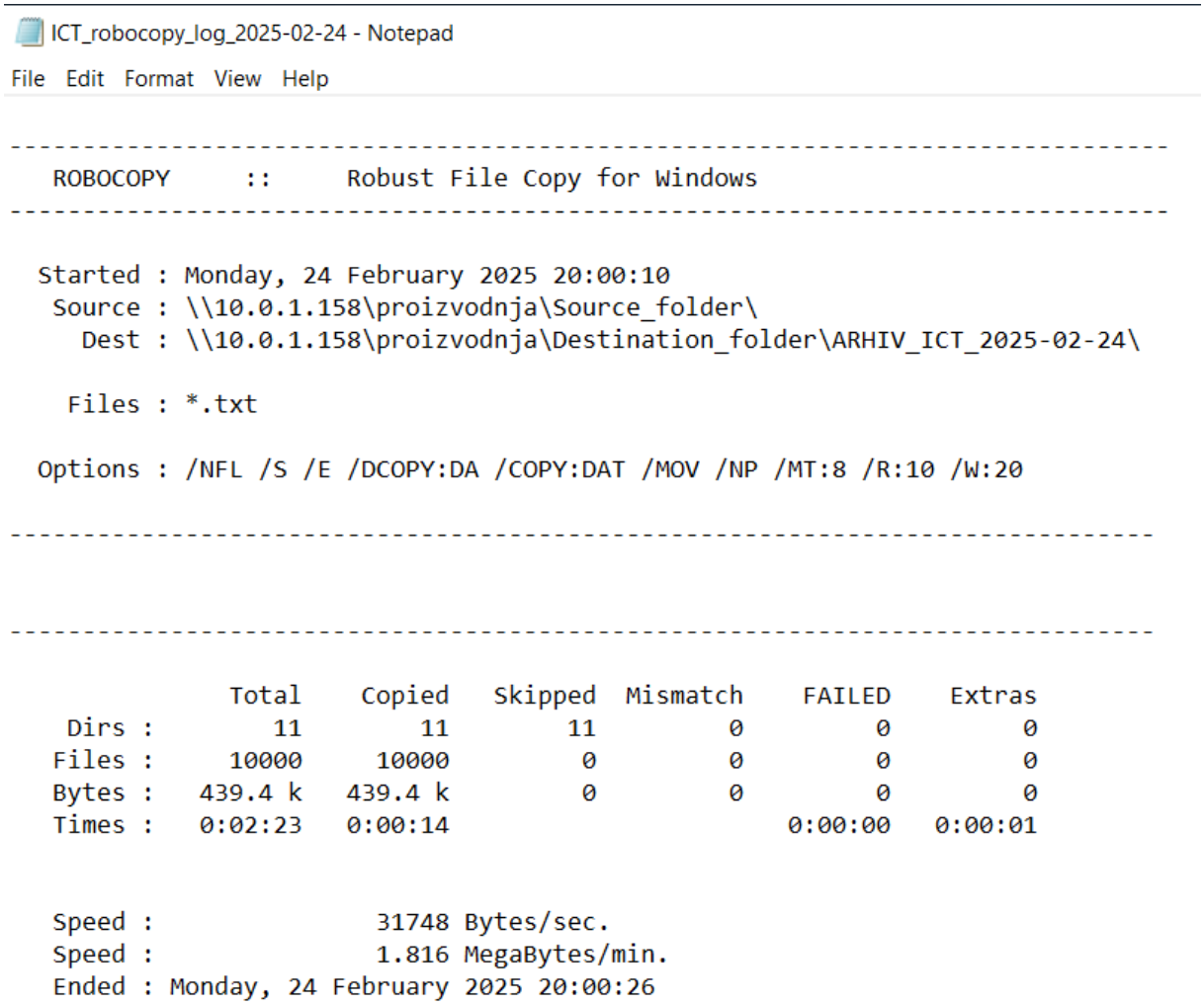
Slika 27: Lastnosti map pred premikanjem

Vir: Lasten



Slika 28: Lastnosti map po premikanju

Vir: Lasten



ICT_robocopy_log_2025-02-24 - Notepad

File Edit Format View Help

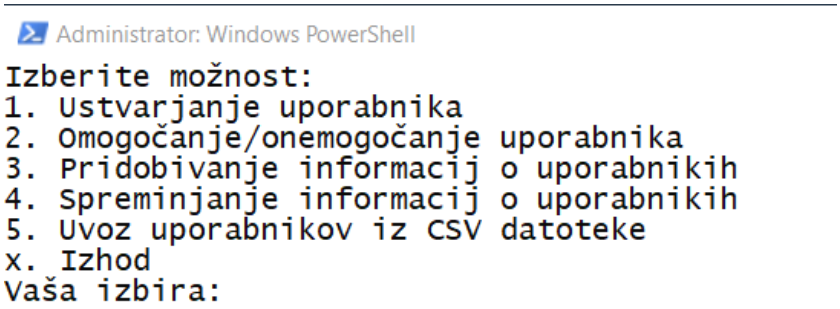
```
-------------------------------------------------------------------------------
   ROBOCOPY     ::     Robust File Copy for Windows
-------------------------------------------------------------------------------

  Started : Monday, 24 February 2025 20:00:10
   Source : \\10.0.1.158\proizvodnja\Source_folder\
     Dest : \\10.0.1.158\proizvodnja\Destination_folder\ARHIV_ICT_2025-02-24\

    Files : *.txt

  Options : /NFL /S /E /DCOPY:DA /COPY:DAT /MOV /NP /MT:8 /R:10 /W:20

------------------------------------------------------------------------------


------------------------------------------------------------------------------

               Total    Copied   Skipped  Mismatch    FAILED    Extras
    Dirs :        11        11        11         0         0         0
   Files :     10000     10000         0         0         0         0
   Bytes :   439.4 k   439.4 k         0         0         0         0
   Times :   0:02:23   0:00:14                       0:00:00   0:00:01


   Speed :               31748 Bytes/sec.
   Speed :               1.816 MegaBytes/min.
   Ended : Monday, 24 February 2025 20:00:26
```

Slika 29: Vsebina dnevniške datoteke robocopy po premikanju

Vir: Lasten

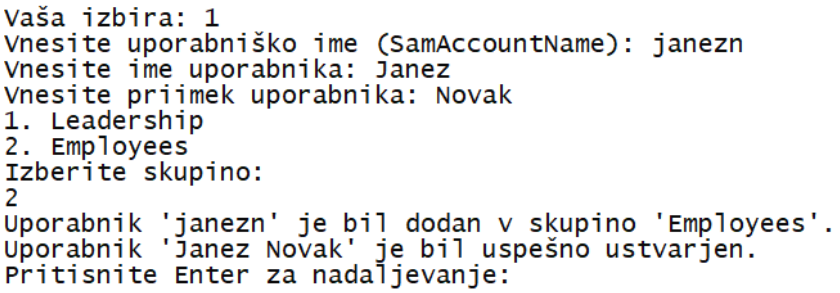
3.1.3 Izdelek – program AD-Toolkit v PowerShellu

V prilogi diplomskega dela je skripta preprostega PowerShell programa »AD-Toolkit.ps1", s katerim lahko uporabnik ureja določene vidike v Active Directoryju. S tem programom lahko kreira nov uporabniški račun, omogoči ali onemogoči že kreiran uporabniški račun, izpiše osnovne podatke o specifičnem uporabniškem računu glede na izbran filter, spreminja informacije o oddelku in nazivu uporabnika ter uvozi datoteko CSV, s katero lahko kreira večje število uporabnikov naenkrat.



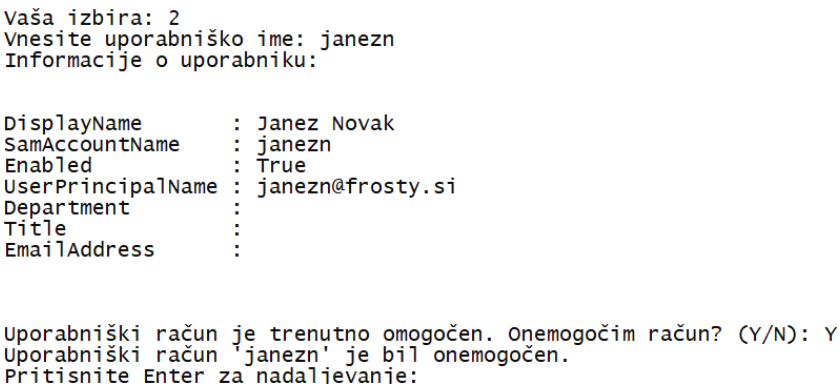
Slika 30: Osnovni prikaz AD-Toolkit

Vir: Lasten



Slika 31: Kreiranje novega uporabnika v AD-Toolkit

Vir: Lasten



Slika 32: Onemogočanje uporabnika v AD-Toolkit

Vir: Lasten

```
Vaša izbira: 3
Izberite skupino (Employees/Leadership/Pustite prazno za vse): Employee
Izberite oddelek (Finance/Marketing/HR/Sales/IT/Pustite prazno za vse):
Uporabniki:

DisplayName   SamAccountName Enabled Department Title
-----------   -------------- ------- ---------- -----
              Guest            False
Ana Novak     anan              True Finance
Luka Horvat   lukah             True Finance    Head of Finance
Maja Kovačič  majak             True Sales
Peter Krajnc  peterk           False
Nina Zupančič ninaz             True Marketing  Head of Marketing
Marko Vidmar  markov            True Sales
Jan Kos       jank              True Sales      Head of Sales
Sara Mlakar   saram             True Sales
Maja Kavčič   majak1            True Sales
David Golob   davidg            True Finance
Igor Medved   igorm             True Finance
Simon Belec   simonb           False
Neža Zajc     nezaz             True Marketing
Urša Kavčič   ursak             True Sales
Matic Rojc    maticr            True IT
Tanja Hribar  tanjah            True Marketing
Domen Šuštar  domens            True IT
Karin Knez    karink            True HR
Jasmina Čeh   jasminac          True HR         Head of HR
Gašper Novak  gaspern           True IT
Pia Kos       piak              True Sales
Teo Krajnc    teok              True IT
Tadej Vidmar  tadejv            True IT         Head of IT
Janez Novak   janezn           False

Pritisnite Enter za nadaljevanje:
```

Slika 33: Izpis podatkov o uporabnikih v AD-Toolkit

Vir: Lasten

```
Vaša izbira: 4
Vnesite uporabniško ime (SamAccountName): janezn

Department:
Job Title:

Izberite oddelek:
1. Finance
2. Marketing
3. HR
4. Sales
5. IT
x. Odstrani oddelek
0. Pustite nespremenjeno
Vaša izbira: 4
Vnesite nov naziv, pustite prazno, če ga ne želite spremeniti, ali vnesite 'x' za odstranitev naziva:

Atributi so bili uspešno spremenjeni.
Pritisnite Enter za nadaljevanje:
```

Slika 34: Urejanje informacij o uporabnikih v AD-Toolkit

Vir: Lasten

```
Vaša izbira: 5
Uporabnik 'markok' je bil uspešno obdelan.
Uporabnik 'lukau' je bil uspešno obdelan.
Pritisnite Enter za nadaljevanje:
```

Slika 35: Uvoz iz datoteke CSV v AD-Toolkit

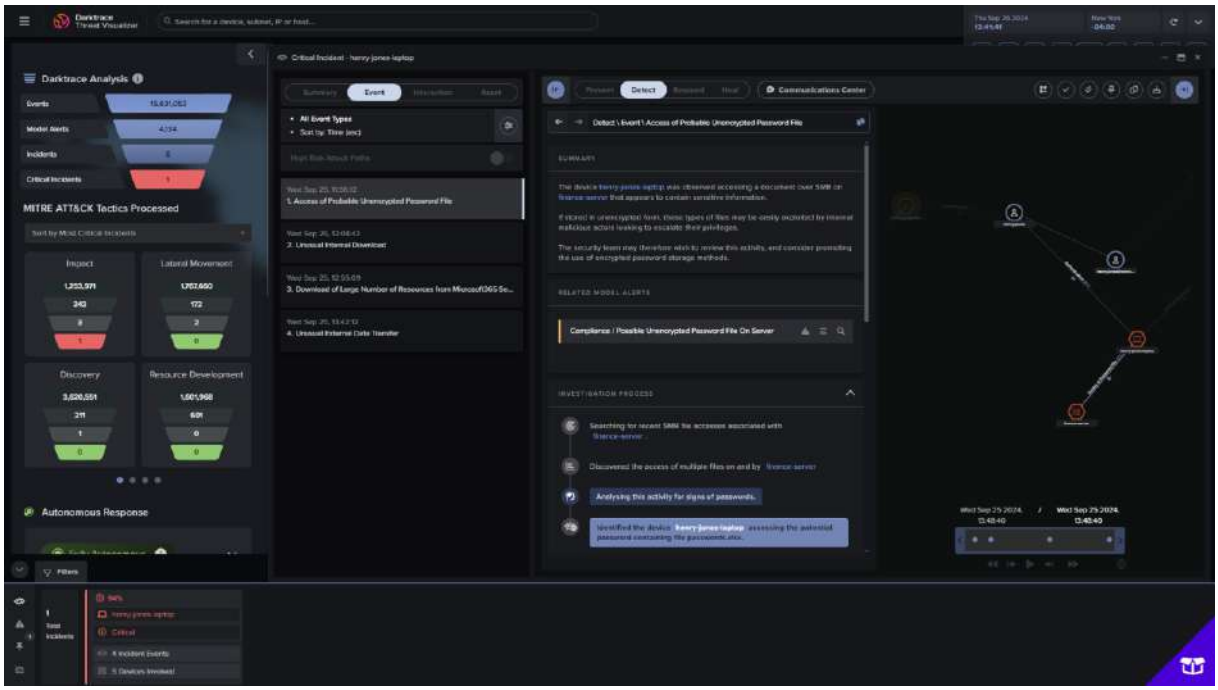
Vir: Lasten

Zgornje slike prikazujejo delovanje PowerShell skripte AD-Toolkit, s katero lahko do določene mere avtomatiziramo dodajanje novih uporabnikov v Active Directory, omogoča pa nam tudi omogočanje in onemogočanje uporabnikov, izpis informacij o uporabnikih in spreminjanje informacij o uporabnikih.

3.2 Primeri integracije umetne inteligence v avtomatizacijo

Uporaba umetne inteligence lahko močno olajša delo sistemskih administratorjev že od samega začetka njihove karierne poti. Prva »postaja« sistemskih administratorjev je tako imenovani »helpdesk« oziroma prva podpora strankam in uporabnikom. Primer uporabe umetne inteligence v take namene je na primer IBM watsonx Assistant – izdelek, ki ga lahko uporabimo za pogovor s stranko/uporabnikom, ki lahko na podlagi baze znanja svetuje in pomaga pri enostavnih težavah, kot so zamenjava gesla pred potekom le-tega, reševanje težav pri tiskanju in podobno. Bistvena prednost takega orodja je v tem, da je dosegljivo 24 ur na dan, 7 dni v tednu, brez čakalnih vrst, če infrastruktura to podpira. (Watsonx Assistant, 2025)

Še eno orodje, ki za svoje delovanje uporablja umetno inteligenco, je Darktrace. Darktrace je orodje, ki uporablja strojno učenje za analizo ogromne količine podatkov o omrežnem prometu, na podlagi tega pa si kreira neko podobo normalnega stanja in obnašanja naprav v omrežju. V naslednjem koraku, če orodje zazna vedenje, ki odstopa od normalnega vzorca, sproži opozorilo, ki ga nato posreduje administratorjem. (Darktrace - Network, 2025) Kompatibilen je tudi z orodjem QRadar, ki je orodje oziroma platforma za varnostno analizo.



Slika 36: Prikaz orodja Darktrace

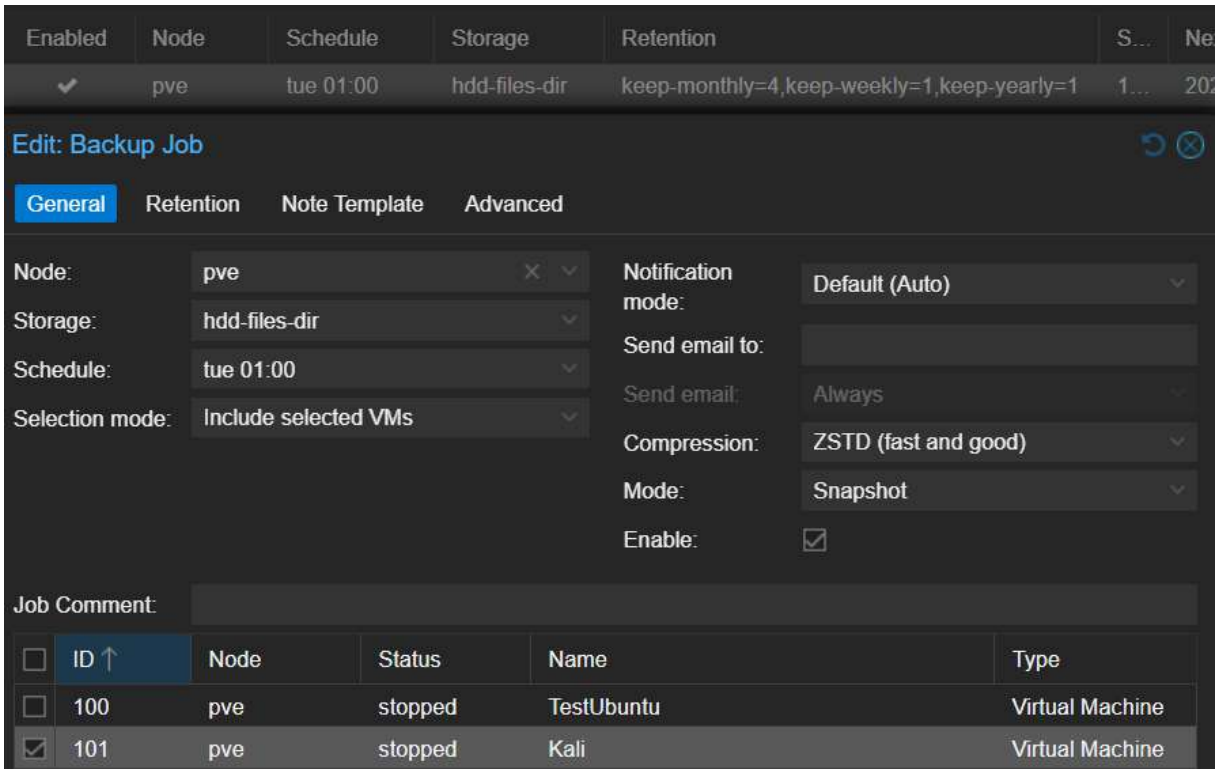
Vir: https://cdn.prod.website-files.com/626ff19cdd07d1258d49238d/67865d9beae5d46f856f5397_Homepage%20w-Highlighted%20Path.png

Tako orodje lahko sistemskim administratorjem zelo olajša delo, seveda le v primeru pravilne vpeljave v IT-infrastrukturo podjetja, saj je bistveno pri zagotavljanju varnosti IT-okolja podjetja.

3.3 Avtomatizacija varnostnih kopij

V sodobni infrastrukturi podjetja dandanes vidimo vedno več uporabe virtualizacije. Kjer virtualni stroji in kontejnerji gostijo kritične podatke in aplikacije, so varnostne kopije nepogrešljiv element zanesljive IT-infrastrukture. Izguba teh sredstev, bodisi zaradi odpovedi strojne opreme, napak v programski opremi, človeških napak ali vse pogostejših zlonamernih napadov, lahko povzroči znatno škodo. Večina hipervizorjev že sama po sebi omogoča kreiranje varnostnih kopij, prav tako pa marsikateri omogoča tudi integracijo namenske opreme za varnostno kopiranje podatkov, med katere spadata Veeam in Rubrik.

Za potrebe primera prikaza avtomatizacije varnostnih kopij smo uporabili hipervizor tipa 1 Proxmox, ki sam po sebi nudi osnovno funkcionalnost varnostnega kopiranja, ki pa se ob uporabi Proxmox Backup Serverja še znatno poveča.



Slika 37: Konfiguracija avtomatizacije varnostnih kopij

Vir: Lasten

Kot je razvidno iz zgornje slike, smo kreirali novo opravilo za avtomatizacijo varnostnih kopij, kjer smo določili shrambo, urnik in način selekcije, izbrali, kateri virtualni stroj želimo varnostno kopirati, prav tako pa smo določili, koliko varnostnih kopij naj program ohrani.

Slika 38: Rezultat avtomatizacije varnostnega kopiranja

Vir: Lasten

Na zgornji sliki je viden rezultat konfiguracije avtomatizacije varnostnega kopiranja – ob vnaprej določeni uri se je ustvarila enostavna varnostna kopija izbranega virtualnega stroja. Ker smo za varnostno kopiranje uporabili le Proxmox Virtual Environment brez Proxmox Backup Serverja, se je ustvarila celotna varnostna kopija virtualnega stroja. V primeru uporabe Proxmox Backup Serverja pa bi lahko vzpostavili kreiranje inkrementalnih varnostnih kopij, kjer bi se po prvi polni varnostni kopiji varnostno kopirali le tisti podatki, ki so se od prejšnje različice spremenili.

3.4 Avtomatizacija posodabljanja programske opreme

Posodabljanje programske opreme je ena ključnih nalog sistemskih administratorjev, saj s tem zagotovimo, da so morebitne varnostne luknje zastarele programske opreme pokrpane. Ko govorimo o IT-okolju, ki šteje več deset ali celo več sto strežnikov, pa se soočimo s časovnimi težavami. Posodabljanje operacijskih sistemov in programske opreme na takšni količini naprav je lahko zelo zamudno, vendar obstajajo orodja, kako sistemski administratorji vse to poenostavijo. Eno teh orodij smo že omenili – to je Ansible, s katerim lahko naenkrat pošljemo ukaze na veliko številno strežnikov naenkrat ter s tem poskrbimo za hitro in učinkovito posodobitev tako operacijskega sistema kot nameščene programske opreme.

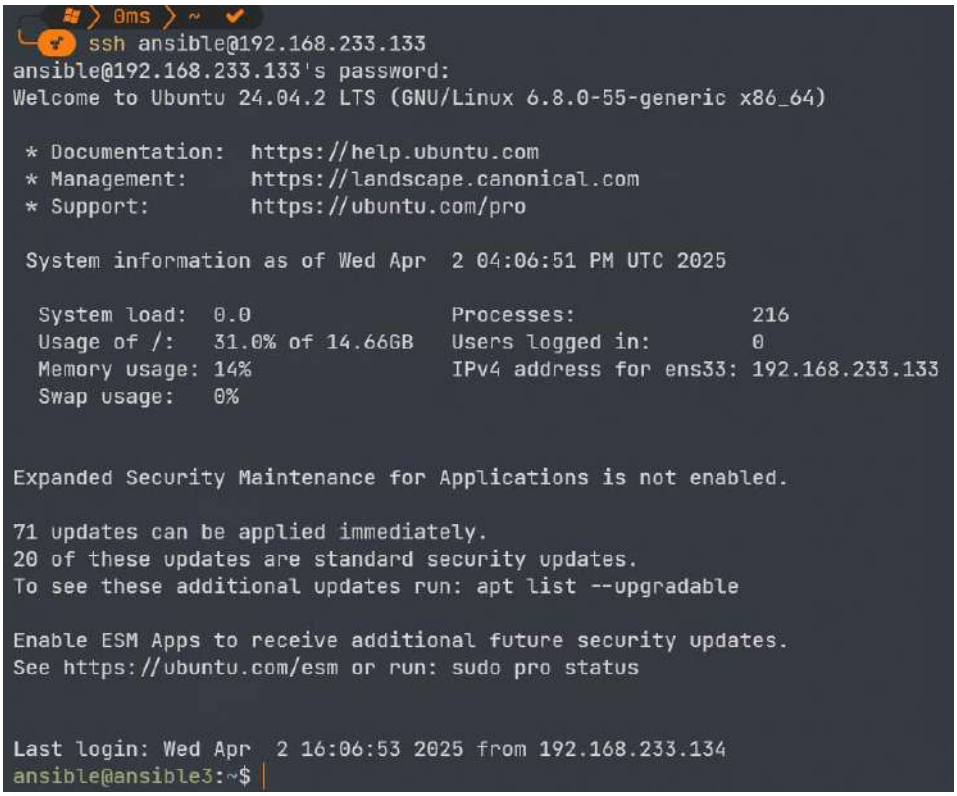
Kot primer smo pripravili Playbook za Ansible z imenom update_upgrade.yml, ki bo posodobil tri vnaprej pripravljene strežnike, ki so v skupini »linux_servers«, z enim samim ukazom.

```
ansible@ansiblemain:~/playbooks$ cat update_upgrade.yml
---
- hosts: linux_servers
  become: yes
  tasks:
    - name: Update apt cache
      apt:
        update_cache: yes

    - name: Upgrade all packages
      apt:
        upgrade: yes
```

Slika 39: Ansible update in upgrade

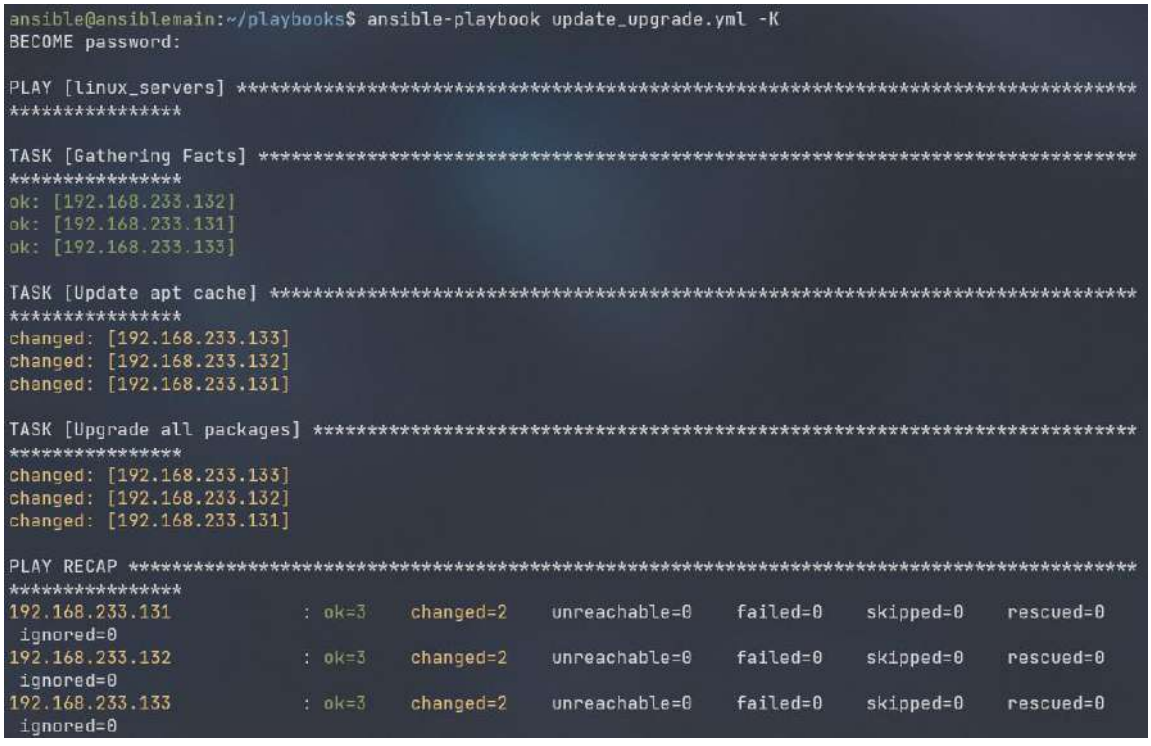
Vir: Lasten

Za zagon tega Playbooka moramo na strežniku, kjer je nameščen Ansible, zagnati ukaz »ansible-playbook update_upgrade.yml -K«. Na spodnjih slikah je možno videti stanje možnih posodobitev na strežniku pred zagonom Playbooka, sam Ansible izpis ob zagonu Playbooka ter stanje možnih posodobitev na strežniku po zagonu Playbooka.



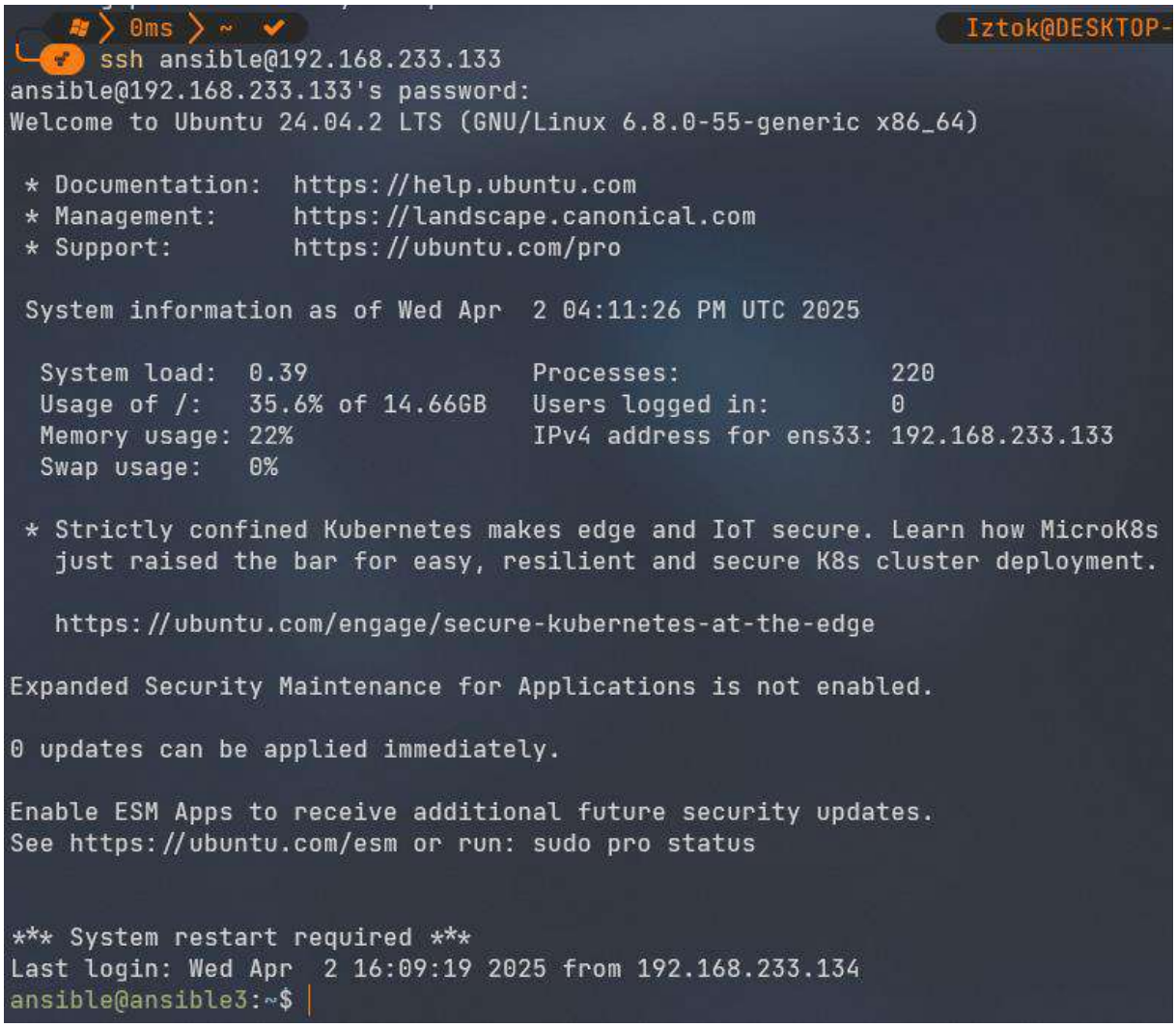
Slika 40: Stanje strežnika pred zagonom Playbooka

Vir: Lasten



Slika 41: Stanje ob zagonu Playbooka

Vir: Lasten



Slika 42: Stanje strežnika po zagonu Playbooka

Vir: Lasten

Rezultat tega primera je, da smo z uporabo enega ukaza in vnaprej pripravljenega Playbooka istočasno zagnali posodabljanje sistemske in programske opreme na treh strežnikih. S tem smo prihranili dragocen čas, ki bi ga zapravili ob ročnem posodabljanju vseh treh strežnikov posebej.

4 SKLEP

Med pisanjem diplomskega dela smo prišli do določenih ugotovitev pri vseh sedmih hipotezah – pri nekaterih skozi teorijo in že dokazanih primerov, pri drugih pa skozi praktične teste na lastnih virtualnih strojih. Sledijo zastavljene hipoteze in potrditve ali zavrnitev le-teh.

Hipoteza 1:

Avtomatizacija varnostnega kopiranja podatkov bistveno zmanjša tveganje izgube podatkov v primeru sistemskih napak.

Prvo hipotezo smo potrdili – to smo dokazali z implementacijo in preizkusom treh različnih avtomatiziranih pristopov k varnostnemu kopiranju. Z uporabo PowerShell skripte (podpodpoglavje **3.1.2)** smo avtomatizirali proces premikanja pomembnih datotek iz delovne mape na namensko arhivsko lokacijo, kar zmanjšuje tveganje izgube podatkov v primeru napake na primarnem mestu shranjevanja. Prav tako smo z uporabo Python skripte (podpodpoglavje **2.2.3)** implementirali rešitev varnostnega kopiranja iz primarne na sekundarno lokacijo. Kot zadnjo implementacijo smo uredili tudi avtomatizacijo kreiranja varnostnih kopij celotnega sistema znotraj virtualizacijskega okolja Proxmox VE. To v praksi pomeni, da bi lahko v primeru sistemske napake na virtualnem stroju hitro obnovili celoten stroj v zadnje delujoče stanje iz shranjene varnostne kopije.

Vsi trije primeri jasno kažejo, da avtomatizacija varnostnega kopiranja prispeva k zmanjšanju tveganja izgube podatkov.

Hipoteza 2:

Avtomatizacija posodabljanja sistema in programske opreme zmanjša število varnostnih ranljivosti v informacijskem sistemu.

Drugo hipotezo smo potrdili na podlagi praktičnega preizkusa z uporabo orodja Ansible (podpoglavje **3.4**) na treh virtualnih strojih. Pred izvedbo avtomatiziranega procesa posodabljanja je bilo na te stroje možno namestiti 71 posodobitev, od tega 20 varnostnih popravkov. Vsak tak nenameščeni popravek predstavlja varnostno ranljivost, ki bi jo lahko izkoristili potencialni napadalci. Razvili smo enostaven Ansible Playbook, prilagojen za avtomatizirano nameščanje vseh čakajočih posodobitev na vnaprej definiranih virtualnih strojih, ki smo ga kasneje tudi zagnali. Po pregledu stanja na omenjenih virtualnih strojih smo ugotovili, da je bilo nameščenih vseh 71 popravkov. Ta rezultat dokazuje veljavnost hipoteze,

saj smo z avtomatizacijo namestitve varnostnih popravkov učinkovito odpravili specifične, znane varnostne pomanjkljivosti, ki so jih ti popravki naslavljali. Z avtomatizacijo procesa posodabljanja zagotovimo, da bodo sistemi dosledno posodobljeni z najnovejšimi varnostnimi popravki, kar neposredno zmanjšuje število obstoječih varnostnih ranljivosti v sistemu.

Hipoteza 3:

Samodejno čiščenje začasnih datotek vodi k znatnemu izboljšanju zmogljivosti sistema in zmanjšanju porabe diskovnega prostora.

Tudi tretjo hipotezo smo potrdili, za kar smo razvili Shell skripto (primer v podpodpoglavju **2.2.1)**, ki ob pomoči pravilno konfiguriranega cronjoba skrbi za brisanje vseh datotek v vnaprej določenih mapah v rednih časovnih intervalih. S tem smo zagotovili manjšo porabo diskovnega prostora, posledično tudi izboljšanje delovanja sistema, prav tako pa na tak način skrbimo za higieno in organizacijo v mapi. Čeprav v tem delu nismo izvajali meritev zmogljivosti sistema pred in po čiščenju, je vseeno splošno sprejeto dejstvo, da lahko prekomerno kopičenje začasnih datotek negativno vpliva na sistemsko zmogljivost. Zaključimo lahko, da avtomatizirano čiščenje začasnih datotek neposredno prispeva k manjši porabi diskovnega prostora in hkrati pozitivno vpliva na ohranjanje oziroma izboljšanje zmogljivosti sistema.

Hipoteza 4:

Spremljanje sistemskih virov in samodejno obveščanje o preseženih pragovih zmanjša odzivni čas na sistemske težave.

Za dokazovanje in potrditev te hipoteze smo v testnem okolju implementirali sistem za nadzor CheckMK (podpodpoglavje **2.2.5)**. V sistem je bilo vključeno spremljanje treh testnih strežnikov, spremljali pa smo CPU Load in dosegljivost strežnika. Ko je sistem CheckMK zaznal mejne vrednosti, je to tudi ustrezno signaliziral v svojem vmesniku. Čeprav v našem okolju ni bilo možno v celoti konfigurirati pošiljanja e-poštnih sporočil, smo ta ključni vidik samodejnega obveščanja uspešno ponazorili s praktičnimi primeri iz delovnega okolja. S pridobljenim soglasjem smo prikazali primere dejanskih e-poštnih obvestil, ki jih CheckMK ob pravilni konfiguraciji pošlje sistemskim administratorjem ob preseganju nastavljenih pragov. Pridobili smo tudi soglasje za prikaz primera osnovnega vmesnika CheckMK, ki prikazuje stanje določenega dela IT-infrastrukture podjetja.

Hipoteza 5:

Avtomatizacija upravljanja uporabniških računov vodi do večje varnostne skladnosti in manj napak v upravljanju pravic dostopa.

Hipoteze v okviru praktičnega dela tega diplomskega dela zaradi omejitev strojne in programske opreme nismo mogli neposredno preizkusiti z implementacijo demonstracijskega primera, vendar pa smo hipotezo potrdili na podlagi teoretičnega pregleda relevantne literature in uveljavljenih praks na področju informacijske varnosti in sistemske administracije. V teoretičnem delu smo izpostavili, da v kompleksnih okoljih ročno upravljanje uporabniških računov postane zamudno, neučinkovito in predvsem podvrženo človeškim napakam. Teoretične ugotovitve kažejo, da ravno človeški dejavnik predstavlja največje tveganje za varnostno skladnost. Pravilno zastavljena avtomatizacija, na primer z uporabo rešitev za upravljanje identitet in dostopa, pa te težave bistveno zmanjšuje.

Hipoteza 6:

Vpeljava avtomatizacije obnovitve po napakah ne izboljša časa obnovitve sistema v primerjavi z ročnimi postopki.

Hipotezo smo na podlagi naših ugotovitev s praktičnim primerom avtomatizacije varnostnih kopij zavrgli. Naš primer jasno nakazuje, da avtomatizirani oziroma delno avtomatizirani postopki obnove bistveno pripomorejo k skrajšanju časa, potrebnega za ponovno vzpostavitev delovanja sistema. Ročna obnova je dolgotrajen proces, ki lahko vključuje ponovno namestitev operacijskega sistema in potrebnih aplikacij, konfiguracijo nastavitev ter obnovo podatkov. V nasprotju s tem pa možnost hitre obnove celotnega virtualnega stroja iz popolne varnostne kopije, ki jo omogoča Proxmox, predstavlja bistveno izboljšavo. Čeprav morda sam sprožilec obnove ni vedno popolnoma avtomatiziran brez človeške intervencije, pa že sama avtomatizacija kreiranja konsistentnih varnostnih kopij in možnost hitre povrnitve sistema v delujoče stanje predstavljata bistveno izboljšavo v času obnovitve. Pomembno je tudi poudariti, da mora biti pri načrtovanju avtomatizacije obnove po napakah pozornost namenjena tudi varnostnim in kontrolnim mehanizmom, saj nočemo, da se ti po nepotrebnem sprožijo zaradi morebitnih kratkotrajnih motenj.

Hipoteza 7:

Uporaba umetne inteligence pri avtomatizaciji sistemske administracije poveča učinkovitost pri odkrivanju in odpravljanju napak.

Hipoteze v okviru praktičnega dela tega diplomskega dela nismo mogli neposredno preizkusiti z lastno implementacijo rešitev zaradi sistemskih in programskih omejitev. Kljub temu pa lahko na podlagi teoretičnega dela relevantne literature hipotezo potrdimo. V sklopu tega smo raziskali orodje Darktrace, ki po javno dostopnih informacijah s strani proizvajalca uporablja umetno inteligenco in strojno učenje za učenje normalnega obnašanja omrežja in sistemov. Na podlagi naučenega pa naj bi Darktrace sistemske administratorje kasneje opozarjal na morebitne zaznane anomalije, kibernetske napade in ostale grožnje v realnem času. Takšen pristop neposredno vpliva na zmanjšanje časa, ki je potreben za odkrivanje in odpravljanje morebitnih napak in incidentov.

Prihodnost sistemske administracije, kot smo ugotovili v diplomskem delu, ni v izginotju vloge, temveč v njeni preobrazbi, ki jo poganjajo trije ključni dejavniki: napredna avtomatizacija, umetna inteligenca in hibridna oblačna okolja. Sistemskemu administratorju bo pri delu v neizogibno pomoč umetna inteligenca, ki ne bo več samo zaznavala anomalij, ampak bo napovedovala možne okvare, avtomatsko določala vzroke težav in do neke mere samostojno izvajala ukrepe. Sistemski administrator bo novo vlogo opravljal v vse bolj kompleksnih hibridnih okoljih, saj trendi kažejo, da prihodnost ni zgolj v javnem oblaku, temveč v premišljeni kombinaciji lokalnih in oblačnih virov. Posledično bodo znanja avtomatizacije, orkestracije med raznimi platformami in optimizacije stroškov ključne veščine, ki definirajo uspešnega sistemskega administratorja prihodnosti.

5 LITERATURA

Anželj, G., Brank, J., Brodnik, A., Bulić, P., Ciglarič, M., Đukić, M., . . . Sterle, P. (2. februar 2025). *E-Učbenik za informatiko v gimnaziji*. Pridobljeno iz https://lusy.fri.uni-lj.si/ucbenik/book/index.html

Barrett, R., Kandogan, E., Maglio, P. P., Haber, E. M., Takayama, L., & Prabaker, M. (2004). Field studies of computer system administrators: Analysis of system management tools and practices. *Proceedings of the 2004 ACM Conference on Computer Supported Cooperative Work*, (str. 388-395). Chicago.

Chapter 1 - Getting started with PowerShell. (3. marec 2025). Pridobljeno iz Microsoft Learn: https://learn.microsoft.com/en-us/powershell/scripting/learn/ps101/01-getting-started?view=powershell-7.5

Charles Babbage. (2. februar 2025). Pridobljeno iz Britannica: https://www.britannica.com/biography/Charles-Babbage

Checkmk 2.3. (19. marec 2025). Pridobljeno iz checkmk: https://checkmk.com/product/latest-version

Darktrace - Network. (18. marec 2025). Pridobljeno iz Darktrace: https://darktrace.com/products/network

Deep Learning in Cybersecurity: Threat Detection and Defense. (20. maj 2025). Pridobljeno iz xenonstack.com: https://www.xenonstack.com/blog/deep-learning-in-cybersecurity

Eniac. (2. februar 2025). Pridobljeno iz Britannica: https://www.britannica.com/technology/ENIAC

How Automation Simplifies User Access Reviews for Remote and Hybrid Workforces. (25. maj 2025). Pridobljeno iz securends.com: https://www.securends.com/blog/automate-user-access-reviews/

Introduction to Ansible. (1. april 2025). Pridobljeno iz ansible.com: https://docs.ansible.com/ansible/latest/getting_started/introduction.html

Fister, I. (2. 11 2023). Sistemska administracija v Linuxu. Univerzitetna založba Univerze v Mariboru.

Limoncelli, T. A., Hogan, C. J., & Chalup, S. R. (2017). *The Practice of System and Network Administration.* Addison Wesley.

Nemeth, E., Snyder, G., Hein, T. R., & Whaley, B. (2010). *UNIX and Linux System Administration Handbook.* Boston: Prentice Hall.

PowerShell nadomešča ukazni poziv. (3. marec 2025). Pridobljeno iz Microsoft Support: https://support.microsoft.com/sl-si/windows/powershell-nadome%C5%A1%C4%8Da-ukazni-poziv-fdb690cf-876c-d866-2124-21b6fb29a45f

Susnjara, S., & Smalley, I. (10. februar 2025). *What is a mainframe?* Pridobljeno iz IBM: https://www.ibm.com/think/topics/mainframe

Turing Machine. (2. Februar 2025). Pridobljeno iz Britannica: https://www.britannica.com/technology/Turing-machine

unix.org. (12. februar 2025). Pridobljeno iz History and Timeline: https://unix.org/what_is_unix/history_timeline.html

Watsonx Assistant. (18. marec 2025). Pridobljeno iz IBM: https://www.ibm.com/products/watsonx-assistant

What Does a Systems Administrator Do? (3. Februar 2025). Pridobljeno iz Florida National University: https://www.fnu.edu/what-does-a-systems-administrator-do/

What is AIOps? (22. maj 2025). Pridobljeno iz aws.amazon.com: https://aws.amazon.com/what-is/aiops/

What is Artificial Intelligence (AI)? (20. maj 2025). Pridobljeno iz cloud.google.com: https://cloud.google.com/learn/what-is-artificial-intelligence

What is PowerShell? (3. marec 2025). Pridobljeno iz Microsoft Learn: https://learn.microsoft.com/en-us/powershell/scripting/overview?view=powershell-7.5

6 PRILOGE

Priloga 1: **Izdelek AD-Toolkit.ps1**

```
#
# Simple AD user management script
#
# Avtor: Iztok Hladen
# Datum: 13.03.2025
#
# Import AD module
Import-Module ActiveDirectory

# Funkcija za ustvarjanje uporabnika
function Create-ADUser {
    # Pridobivanje uporabniškega imena in preverjanje zasedenosti
    do {
        $Username = Read-Host "Vnesite uporabniško ime (SamAccountName)"
        if (Get-ADUser -Filter "SamAccountName -eq '$Username'") {
            Write-Host "Uporabniško ime '$Username' je že zasedeno. Izberite
drugo."
        }
    } while (Get-ADUser -Filter "SamAccountName -eq '$Username'")

    # Pridobivanje imena in priimka
    $Ime = Read-Host "Vnesite ime uporabnika"
    $Priimek = Read-Host "Vnesite priimek uporabnika"

    # Izbira skupine
    Write-Host "1. Leadership"
    Write-Host "2. Employees"
    Write-Host "Izberite skupino:"
    $GroupChoice = Read-Host


    # Določanje skupine
    if ($GroupChoice -eq "1") {
        $GroupDN = "CN=Leadership,CN=Users,DC=frosty,DC=si"
    } elseif ($GroupChoice -eq "2") {
        $GroupDN = "CN=Employees,CN=Users,DC=frosty,DC=si"
    } else {
        Write-Host "Napačna izbira. Uporabnik ne bo dodan v nobeno skupino."
    }

    # Ustvarjanje DisplayName
    $displayName = "$Ime $Priimek"

    # Default geslo
    $Password = (ConvertTo-SecureString "FrOsty.01" -AsPlainText -Force)

    # Ustvarjanje AD uporabnika
    New-ADUser
        -SamAccountName $Username `
        -Name $displayName `
        -DisplayName $displayName `
        -GivenName $Ime `
        -Surname $Priimek `
        -UserPrincipalName $Username@frosty.si `
        -AccountPassword $Password `
        -Path "CN=Users,DC=frosty,DC=si" `
        -ChangePasswordAtLogon $true `
        -Enabled $true

    # Dodajanje v skupino
    if ($GroupDN) {
        Add-ADGroupMember -Identity $GroupDN -Members $Username
        Write-Host "Uporabnik '$Username' je bil dodan v skupino
'$($GroupDN.Split(',')[0].Split('=')[1])'."
    }

    Write-Host "Uporabnik '$displayName' je bil uspešno ustvarjen."
}
```

```powershell
# Funkcija za omogočanje/onemogočanje uporabnika
function Enable-Disable-ADUser {
    # Pridobivanje uporabniškega imena
    $Username = Read-Host "Vnesite uporabniško ime"

    # Pridobivanje informacij o uporabniku
    $User = Get-ADUser -Identity $Username -Properties Enabled, DisplayName,
SamAccountName, UserPrincipalName, Department, Title, EmailAddress

    # Preverjanje, ali je uporabnik najden
    if ($User) {
        # Izpis informacij o uporabniku
        Write-Host "Informacije o uporabniku:"
        $User | Format-List DisplayName, SamAccountName, Enabled,
UserPrincipalName, Department, Title, EmailAddress

        # Vprašanje o omogočanju/onemogočanju računa
        if ($User.Enabled) {
            do {
                $Action = Read-Host "Uporabniški račun je trenutno omogočen.
Onemogočim račun? (Y/N)"
            } while ($Action -notin ("Y", "N"))

            if ($Action -eq "Y") {
                Disable-ADAccount -Identity $Username
                Write-Host "Uporabniški račun '$Username' je bil onemogočen."
            } else {
                Write-Host "Onemogočanje uporabniškega računa je bilo preklicano."
            }
        } else {
            do {
                $Action = Read-Host "Uporabniški račun je trenutno onemogočen.
Omogočim račun? (Y/N)"
            } while ($Action -notin ("Y", "N"))

            if ($Action -eq "Y") {
                Enable-ADAccount -Identity $Username
                Write-Host "Uporabniški račun '$Username' je bil omogočen."
            } else {
                Write-Host "Omogočanje uporabniškega računa je bilo preklicano."
            }
        }
    } else {
        Write-Host "Uporabnik '$Username' ni bil najden."
    }
}

# Funkcija za izpis podatkov o uporabnikih
function Get-User-Info {
    # Izbira skupine (prazno za vse skupine)
    $GroupChoice = Read-Host "Izberite skupino (Employees/Leadership/Pustite prazno
za vse)"
    while ($GroupChoice -notin ("Employees", "Leadership") -and $GroupChoice -ne
"") {
        Write-Host "Napačna izbira. Izberite Employees, Leadership ali pustite
prazno."
        $GroupChoice = Read-Host "Izberite skupino (Employees/Leadership/Pustite
prazno za vse)"
    }

    # Izbira oddelka (prazno za vse oddelke)
    $DepartmentChoice = Read-Host "Izberite oddelek
(Finance/Marketing/HR/Sales/IT/Pustite prazno za vse)"
    while ($DepartmentChoice -notin ("Finance", "Marketing", "HR", "Sales", "IT") -
and $DepartmentChoice -ne "") {
        Write-Host "Napačna izbira. Izberite Finance, Marketing, HR, Sales, IT ali
pustite prazno."
        $DepartmentChoice = Read-Host "Izberite oddelek
(Finance/Marketing/HR/Sales/IT/Pustite prazno za vse)"
    }

    # Določanje skupin
    $LeadershipDN = "CN=Leadership,CN=Users,DC=frosty,DC=si"
    $EmployeesDN = "CN=Employees,CN=Users,DC=frosty,DC=si"

    # Pridobivanje vseh uporabnikov
```

```powershell
    $Users = Get-ADUser -Filter * -Properties DisplayName, Department, Enabled,
DistinguishedName, MemberOf, Title

    # Filtriranje
    $FilteredUsers = @()

    foreach ($User in $Users) {
        $GroupMatch = $true
        $DepartmentMatch = $true

        # Preverjanje pripadnosti skupini
        if ($GroupChoice -eq "Leadership") {
            $GroupMatch = ($User.MemberOf -contains $LeadershipDN)
        } elseif ($GroupChoice -eq "Employees") {
            $GroupMatch = ($User.MemberOf -contains $EmployeesDN)
        }

        # Preverjanje oddelka
        if ($DepartmentChoice -ne "") {
            $DepartmentMatch = ($User.Department -eq $DepartmentChoice)
        }

        # Če oba kriterija ustrezata, dodamo uporabnika v seznam
        if ($GroupMatch -and $DepartmentMatch) {
            $FilteredUsers += $User
        }
    }

    # Izpis uporabnikov
    if ($FilteredUsers.Count -gt 0) {
        Write-Host "Uporabniki:"
        $FilteredUsers | Format-Table DisplayName, SamAccountName, Enabled,
Department, Title
    } else {
        Write-Host "Ni uporabnikov z izbranimi kriteriji."
    }
}

# Funkcija za spreminjanje oddelka in naziva
function Change-Info {
    # Pridobivanje uporabniškega imena
    $Username = Read-Host "Vnesite uporabniško ime (SamAccountName)"

    # Preverjanje, ali uporabnik obstaja
    if (Get-ADUser -Filter "SamAccountName -eq '$Username'") {

        # Pridobivanje informacij o uporabniku
        $User = Get-ADUser -Identity $Username -Properties Department, Title

        # Izpis trenutnih atributov
        Write-Host
        Write-Host "Department: $($User.Department)"
        Write-Host "Job Title: $($User.Title)"
        Write-Host

        # Meni za izbiro oddelka
        Write-Host "Izberite oddelek:"
        Write-Host "1. Finance"
        Write-Host "2. Marketing"
        Write-Host "3. HR"
        Write-Host "4. Sales"
        Write-Host "5. IT"
        Write-Host "x. Odstrani oddelek"
        Write-Host "0. Pustite nespremenjeno"

        $DepartmentChoice = Read-Host "Vaša izbira"

        # Preverjanje izbire oddelka
        switch ($DepartmentChoice) {
            "1" { $Department = "Finance" }
            "2" { $Department = "Marketing" }
            "3" { $Department = "HR" }
            "4" { $Department = "Sales" }
            "5" { $Department = "IT" }
            "x" { $Department = $null }
            "0" { $Department = $User.Department }
            default {
```

```
                Write-Host "Napačna izbira. Oddelek bo ostal nespremenjen."
                $Department = $User.Department
            }
        }

        # Vnos novih atributov za naziv
        $JobTitle = Read-Host "Vnesite nov naziv, pustite prazno, če ga ne želite
spremeniti, ali vnesite 'x' za odstranitev naziva"

        # Spreminjanje atributov
        try {
            if ($Department -eq $null) {
                Set-ADUser -Identity $Username -Clear Department
            } else {
                Set-ADUser -Identity $Username -Department $Department
            }

            if ($JobTitle -ne "") {
                if ($JobTitle -eq "x") {
                    Set-ADUser -Identity $Username -Clear Title
                } else {
                    Set-ADUser -Identity $Username -Title $JobTitle
                }
            }
            Write-Host "Atributi so bili uspešno spremenjeni."
        } catch {
            Write-Error "Napaka pri spreminjanju atributov:
$($_.Exception.Message)"
        }
    } else {
        Write-Host "Uporabnik '$Username' ni bil najden."
    }

}

# Funkcija za uvoz CSV datoteke ki kreira uporabnike
function Create-ADUser-CSV {
    # Privzeto geslo
    $DefaultPassword = ConvertTo-SecureString "FrOsty.01" -AsPlainText -Force

    # Uvoz CSV datoteke
    $Users = Import-Csv -Path "C:\temp\uporabniki.csv"

    # Obdelava vsakega uporabnika
    foreach ($User in $Users) {
        try {
            # Odstranjevanje skritih znakov in presledkov
            $SamAccountName = $User."uporabniško ime".Trim()
            $Ime = $User.ime.Trim()
            $Priimek = $User.priimek.Trim()
            $Skupina = $User.skupina.Trim()
            $Oddelek = $User.oddelek.Trim()
            $Naziv = $User.naziv.Trim()

            # Ustvarjanje DisplayName in Name
            $DisplayName = "$Ime $Priimek"
            $Name = $DisplayName

            # Ustvarjanje AD uporabnika
            $NewUser = New-ADUser -SamAccountName $SamAccountName `
                                  -Name $Name `
                                  -DisplayName $DisplayName `
                                  -GivenName $Ime `
                                  -Surname $Priimek `
                                  -UserPrincipalName "$SamAccountName@frosty.si"
`
                                  -AccountPassword $DefaultPassword `
                                  -Enabled $true `
                                  -ChangePasswordAtLogon $true `
                                  -Path "CN=Users,DC=frosty,DC=si"

            # Zagotovimo, da je $NewUser pravilno nastavljen
            if (-not $NewUser -and (Get-ADUser -Identity $SamAccountName)) {
                $NewUser = Get-ADUser -Identity $SamAccountName
            }

            # Dodajanje v skupino
```

```
            if ($Skupina -eq "Leadership") {
                Add-ADGroupMember -Identity
"CN=Leadership,CN=Users,DC=frosty,DC=si" -Members $NewUser
            } elseif ($Skupina -eq "Employees") {
                Add-ADGroupMember -Identity "CN=Employees,CN=Users,DC=frosty,DC=si"
-Members $NewUser
            } else {
                Write-Warning "Opozorilo: Nepravilna skupina '$Skupina' za
uporabnika '$SamAccountName'."
            }

            # Nastavitev oddelka in naziva
            if ($NewUser) {
                Set-ADUser -Identity $NewUser -Department $Oddelek -Title $Naziv
            } else {
                Write-Warning "Opozorilo: Uporabnika '$SamAccountName' ni bilo
mogoče najti za nastavitev atributov."
            }

            Write-Host "Uporabnik '$SamAccountName' je bil uspešno obdelan."
        } catch {
            Write-Error "Napaka pri obdelavi uporabnika '$SamAccountName':
$($_.Exception.Message)"
        }
    }

}

# Glavna zanka
do {
    Write-Host "Izberite možnost:"
    Write-Host "1. Ustvarjanje uporabnika"
    Write-Host "2. Omogočanje/onemogočanje uporabnika"
    Write-Host "3. Pridobivanje informacij o uporabnikih"
    Write-Host "4. Spreminjanje informacij o uporabnikih"
    Write-Host "5. Uvoz uporabnikov iz CSV datoteke"
    Write-Host "x. Izhod"

    $Choice = Read-Host "Vaša izbira"

    switch ($Choice) {
        "1" {
            Create-ADUser
        }
        "2" {
            Enable-Disable-ADUser
        }
        "3" {
            Get-User-Info
        }
        "4" {
            Change-Info
        }
        "5" {
            Create-ADUser-CSV
        }
        "x" {
            Write-Host "Izhod iz skripte."
        }
        default {
            Write-Host "Napačna izbira. Poskusite ponovno."
        }
    }

    if ($Choice -ne "x") {
        Read-Host "Pritisnite Enter za nadaljevanje"
    }

} while ($Choice -ne "x")
```