Hochschule für Technik und Wirtschaft Berlin
University of Applied Sciences

Skriptgesteuerte Erstellung und Konfiguration von Windows-basierten virtuellen Maschinen in Hyper-V mittels PowerShell

Bachelorarbeit

von

Kevin Hübner

Matrikelnummer: 570746

Fachbereich 1: Computer Engineering
Hochschule für Technik und Wirtschaft Berlin

Datum: Berlin, 22.08.2025

Erstgutachten:
Zweitgutachten:

Inhaltsverzeichnis

1 Einleitung **2**

2 Praxis **2**

3 Ziel der Arbeit **3**

4 Theorie **3**
4.1 Windows Hyper-V 3
4.2 Windows Powershell 4
4.3 Sysprep 5

5 Entwicklung des Skripts zur Automatisierung **5**
5.1 Vorbereitung des VM-Templates 5
5.2 Erstellung der VMs mittels PowerShell 7
5.3 Verzeichnisstruktur und Logdateien 8
5.4 Konfiguration der virtuellen Maschinen 8

6 Testumgebungen **12**
6.1 Testumgebung Server 2 14

7 Technische Erkenntnisse der Skripterstellung **15**

8 Fazit **18**

1 Einleitung

Mit dem fortschreitenden technologischen Wandel stehen Systemadministratoren eine Vielzahl an Werkzeugen zur Verfügung, um ihre Arbeit effizienter zu gestalten. Dazu gehören grafische Benutzeroberflächen, die die Bedienung vereinfachen, Skripte zur Automatisierung wiederkehrender Prozesse sowie zunehmend auch Methoden der Künstlichen Intelligenz. Besonders kleine und mittelständische Unternehmen sind jedoch häufig damit ausgelastet, ihre bestehenden Systeme im täglichen Betrieb zuverlässig zu betreiben. Dadurch geraten neue Technologien, die die Arbeit erleichtern und langfristig effizienter gestalten könnten, oftmals in den Hintergrund.

2 Praxis

Die Idee zu dieser Arbeit entstand aus praktischen Erfahrungen in der Windows Server-Administration, insbesondere im Umgang mit Hyper-V in Server Umgebungen. In der bisherigen Vorgehensweise erfolgt die Bereitstellung neuer virtueller Maschinen (VMs) vollständig manuell – von der Erstellung der VM über die Installation des Betriebssystems bis hin zur Konfiguration der benötigten Rollen und Dienste.

Ein typisches Szenario ergibt sich bei der Einrichtung eines neuen Kunden im Rechenzentrum. Standardmäßig werden dabei drei VMs eingerichtet: ein Domänencontroller (DC), ein Dateiserver (FS) sowie ein Terminalserver (TS). Diese klare Trennung der Rollen bietet Vorteile hinsichtlich Übersichtlichkeit, Lastverteilung und Sicherheit. Die manuelle Einrichtung umfasst jedoch zahlreiche wiederkehrende Arbeitsschritte, wie die Vergabe statischer IP-Adressen, die Umbenennung der Rechner, die Einbindung zusätzlicher Festplatten, die Erstellung von Standard-Benutzern und -Gruppen sowie die Konfiguration von Gruppenrichtlinien und Ordnerberechtigungen. Hinzu kommt, dass kundenspezifische Anwendungen wie DATEV besondere Anforderungen stellen, beispielsweise durch den Einsatz von SQL-Datenbanken. Diese zusätzlichen Konfigurationen erhöhen den manuellen Aufwand erheblich.

3 Ziel der Arbeit

Um diesen Herausforderungen zu begegnen, wurde im Rahmen dieser Arbeit ein PowerShell-Skript entwickelt, das die wiederkehrenden Arbeitsschritte bei der Bereitstellung von Windows-Server-VMs weitgehend automatisiert. Das Skript übernimmt unter anderem die Erstellung und Konfiguration von drei VMs (DC, FS, TS), die Einrichtung der grundlegenden Rollen, die Anlage von Standard-Benutzern und -Gruppen, die Vergabe von NTFS-Berechtigungen sowie die Konfiguration von Netzwerkparametern.

Als Grundlage dient ein mit dem Windows-Tool Sysprep generalisiertes VM-Template, das mithilfe einer vorbereiteten Antwortdatei (*unattend.xml*) erstellt wird. Dieses Template ermöglicht es, alle weiteren Konfigurationsschritte skriptgesteuert durchzuführen, wodurch einsatzbereite Systeme mit minimalem manuellem Aufwand bereitgestellt werden können.

Ziel dieser Arbeit ist es daher, ein strukturiertes, skriptbasiertes Vorgehen für die automatisierte Erstellung und Konfiguration von Windows-Server-VMs in Hyper-V zu entwickeln und zu evaluieren. Dadurch soll gezeigt werden, wie sich typische Administrationsaufgaben standardisieren und effizienter gestalten lassen.

4 Theorie

4.1 Windows Hyper-V

Hyper-V ist eine unter Windows verfügbare Virtualisierungsplattform. die als optionale Server- bzw. Client-Rolle bereitgestellt werden kann (z.B. in Windows 10/11 Pro oder Windows Server-Versionen). Nach der Installation stellt Hyper-V eine hardwarenahe Virtualisierungstechnologie bereit, die es ermöglicht, mehrere virtuelle Maschinen (VMs) parallel auf einem Hostsystem auszuführen. Dabei können unterschiedliche Betriebssysteme Gastbetriebssysteme installiert werden.
Der Hypervisor von Microsoft ist als sogenannte Typ-1-Hypervisor-Architektur realisiert. Treiber sind dabei nicht direkt im Hypervisor selbst integriert, sondern im Management-Betriebssystem, der sogenannten Parent Partition. Gastbetriebssysteme, die in Child Partitions laufen, kommunizieren mit der Hardware nicht unmittelbar, sondern über den VMBus. Dieser Mechanismus überträgt Ein- und Ausgaben (I/O-Operationen) effizient zwischen Host und

Gast und reduziert potenzielle Treiberkonflikte, da der Hypervisor isoliert unterhalb des Betriebssystems, aber oberhalb der Hardware arbeitet.
Hyper-V unterstützt verschiedene Formate virtueller Festplatten (VHD/VHDX), die entweder mit fester Größe oder dynamisch wachsend bereitgestellt werden können. Dynamische Datenträger bieten den Vorteil, dass der Speicher auch im laufenden Betrieb der virtuellen Maschine erweitert werden kann. Zusätzlich können physische Festplatten direkt eingebunden werden; die dabei auftretenden Performanceverluste sind vergleichsweise gering.[1–3]

4.2 Windows Powershell

Die erste Veröffentlichung von PowerShell erfolgte im Jahr 2006 mit Version 1.0. Mit Windows 10 wurde PowerShell 5.0 standardmäßig in das Betriebssystem integriert, ebenso in Windows Server 2012. Bis heute ist in aktuellen Windows-Versionen, wie Windows 11 sowie Windows Server 2025, PowerShell 5.1 weiterhin als Bestandteil enthalten. Parallel dazu wird die plattformübergreifende Weiterentwicklung unter der Bezeichnung „PowerShell“ (früher „PowerShell Core“) aktiv gepflegt; die derzeit aktuelle Version ist 7.5.2.
PowerShell stellt für Windows-Systeme eine funktionale Entsprechung zur klassischen Shell in Unix/Linux-Umgebungen dar und hat sich in den letzten Jahren von einem Administrationswerkzeug zu einer vollwertigen Automatisierungsplattform entwickelt. Die Befehle, sogenannte „Cmdlets“, folgen einem objektorientierten Ansatz, da sie .NET-Objekte zurückgeben. Dies ermöglicht eine direkte Weiterverarbeitung über die Pipeline, die funktional an Unix-Shells erinnert, jedoch nicht mit Textströmen, sondern mit Objekten arbeitet. Dadurch können Eigenschaften und Methoden von Objekten mit dem Punkt-Operator gezielt angesprochen werden.
PowerShell gilt als Skriptsprache mit besonderem Fokus auf die Automatisierung, Überwachung und Konfiguration von Systemen und wird daher intensiv von Administratoren eingesetzt. Zusätzlich zur lokalen Nutzung unterstützt PowerShell auch die Ausführung von Befehlen auf entfernten Systemen („Remoting“). Dies kann entweder in Form von Einmalbefehlen oder als persistente Remote-Sitzung erfolgen, setzt jedoch gültige Benutzeranmeldedaten auf dem Zielsystem voraus.
Eine besondere Funktion stellt „PowerShell Direct“ dar, die im Kontext von Hyper-V eingesetzt wird. Hierbei kann ein virtueller Computer direkt über den Hyper-V-Host verwaltet werden, ohne dass eine Netzwerkverbindung erforderlich ist. Voraussetzung bleibt auch hier ein gültiger Benutzeraccount auf dem jeweiligen virtuellen System.
Durch diese Eigenschaften vereint PowerShell klassische Skriptmöglichkeiten

mit moderner Objektorientierung und systemübergreifender Automatisierung.[4, 5]

4.3 Sysprep

Sysprep ist ein in Windows integriertes Systemwerkzeug, das sich im Verzeichnis Eine erweiterte Konfiguration ist über sogenannte Unattend-Antwortdateien möglich, wodurch sich bestimmte Einstellungen beim ersten Start automatisch anwenden lassen. Allerdings bestehen auch Einschränkungen: Auf Systemen mit mehreren Betriebssystemen muss für jede Installation separat eine Generalisierung durchgeführt werden. Zudem werden nicht alle Serverrollen durch Sysprep vollständig unterstützt.[6, 7]

5 Entwicklung des Skripts zur Automatisierung

Für die Umsetzung der automatisierten Erstellung und Konfiguration der virtuellen Maschinen wurden verschiedene PowerShell-Module eingesetzt, die spezifische Verwaltungs- und Konfigurationsaufgaben unterstützen. Die grundlegende Funktionalität von PowerShell wird dabei über die Kernmodule *Microsoft.PowerShell.Core*, *Microsoft.PowerShell.Management*, *Microsoft.PowerShell.Security* sowie *Microsoft.PowerShell.Utility* bereitgestellt [8–11]. Darüber hinaus ermöglichen Module wie *NetTCPIP*, *NetAdapter*, *DnsServer* und *DnsClient* die Konfiguration und Verwaltung der Netzwerkumgebung [12–15]. Die Einbindung in eine Active-Directory-Infrastruktur und Gruppenrichtlinien-Infrastruktur erfolgt über die Module *ActiveDirectory* und *GroupPolicy* [16, 17]. Für die Speicher- und Serverkonfiguration wurden die Module *Storage* und
ServerManager verwendet [18, 19]. Die Erstellung und Verwaltung der virtuellen Maschinen selbst basiert auf dem *Hyper−V*-Modul [20]. Zusätzlich kamen Module wie *RemoteDesktop* und verschiedene PowerShell-Beispiele (z. B. zur Erstellung von Eingabedialogen) zum Einsatz [21, 22].

5.1 Vorbereitung des VM-Templates

Im Rahmen der Skripterstellung wurde zunächst ein VM-Template erstellt. Mit dem VM-Template wird die erneute Installation von Windows in den VMs umgangen, womit es nur noch einmal für das Template selbst installiert werden muss. Ein wesentlicher Bestandteil dieses Schrittes war die Erstellung

einer Antwortdatei (Unattend-XML), die im späteren Prozess die automatisierte Erstkonfiguration ermöglicht.

Zur Generierung der Antwortdatei wurde die Windows-Server-ISO eingebunden und mit dem Windows System Image Manager das darin enthaltene Abbild (install.wim) geladen. Anschließend wurde die Zieledition „Server Standard“ ausgewählt, da diese als Grundlage für die Kunden-VMs vorgesehen ist. Vor der Definition der eigentlichen Antwortdatei wurde eine Katalogdatei erzeugt. Innerhalb der Antwortdatei wurden die relevanten Konfigurationsoptionen integriert. Diese sind im Bereich Microsoft-Windows-Shell-Setup zu finden. Dabei lag der Fokus auf dem Abschnitt OOBE, um Ersteinrichtungsoptionen wie Tastaturlayout und Sprache zu überspringen. Zusätzlich wurden noch lokale Benutzerkonten angelegt. Nach Erstellung der XML-Datei wurde diese, der Installation im Audit-Modus hinzugefügt.

Die vollständige XML-Datei wurde im Sysprep Verzeichnis (Sysprep, i.d.R. unter *C:\Windows\System32\Sysprep*) gespeichert. Sysprep ist ein Windows eigenes Tool, welches zur Vorbereitung von Windowssystemen dient. Anschließend erfolgte die Ausführung von Sysprep mit den Parametern */oobe , /generalize, /shutdown /unattend:(Pfad zur XML−Datei).* Dadurch wurde die Windows-Installation generalisiert und von der spezifischen VM Umgebung entkoppelt. Die einmalige Erstellung und Konfiguration beanspruchte auf einem leistungsfähigen System etwa fünf bis zehn Minuten. Das ist die Zeit um Windows-Server auf der VM zu installieren und mit der Tastenkombination STRG+Shift+F3, während der Abfrage des Administratorpassworts, in den Audit-Modus zu wechseln. Dies ermöglichte individuelle Anpassungen sowie die Absicherung des lokalen Administratorkontos während des Bootens in die Windows Umgebung und überspringt die Ersteinrichtung.

Für das Skript war ausschließlich relevant, dass die Passwortvergabe gewährleistet, ein zusätzlicher lokaler Administrator angelegt und Einrichtungsdialoge durch die XML-Datei übersprungen wurden. Ergänzend wurde das Tastaturlayout auf Deutsch gesetzt. Durch die Generalisierung der Windows-Installation wird erreicht, dass die virtuelle Festplatte (VHD/X) einer erstellten VM kopiert und mehrfach wiederverwendet werden kann, ohne dass Konflikte zwischen den Sicherheits-IDs (SIDs) der Administratorbenutzer verschiedener VMs auftreten. Entscheidend ist hierbei, dass die betreffende VM nach der Generalisierung nicht erneut gestartet wird, bevor die VHD(X)-Datei kopiert wurde, da ansonsten die Generalisierung ihre Gültigkeit verliert.

Im Zuge der Experimente mit den VM-Templates zeigte sich jedoch, dass

Fehler in der Generalisierung (z. B. durch fehlerhafte Unattend-Dateien) dazu führen können, dass lokale Administratoren auf unterschiedlichen Systemen identische Sicherheits-IDs (SIDs) erhalten. In diesem Fall schlägt der Domänenbeitritt fehl. Daher ist bei der Erstellung der Unattend-Datei besondere Sorgfalt erforderlich. Die Überprüfung der SIDs kann nach der VM-Erstellung über den Befehl *Get−LocalUser −Name Administrator* FL| erfolgen. Werden identische SIDs festgestellt, ist davon auszugehen, dass die Generalisierung mit Sysprep nicht korrekt durchgeführt wurde, was wiederum zum Abbruch des Skripts führen würde.

5.2 Erstellung der VMs mittels PowerShell

Im nächsten Schritt erfolgte die Erstellung von VMs mithilfe von PowerShell. Der grundlegende Ablauf entspricht dabei den Prozessen, die auch aus anderen Virtualisierungslösungen bekannt sind: Jedem virtuellen System werden ein Name, ein Speicherort, Arbeitsspeicher, CPU-Kerne, ein Netzwerkadapter sowie eine Bootfestplatte zugewiesen. Da bereits eine vorbereitete VHDX-Datei für die Bootpartition vorliegt, muss diese lediglich als Bootmedium zugewiesen werden. Dafür muss die vorbereitete VHD(X)-Datei an den vorgesehenen Speicherort kopiert werden und bei der Erstellung der VM muss der Pfad zu dieser Kopie angegeben werden. Für den Fileserver wurde zusätzlich eine separate Festplatte eingerichtet, die der Speicherung der Nutzerdaten dient und entsprechend in das System eingebunden wird. Die Konfiguration über PowerShell erforderte somit lediglich Schritte, die auch über die grafische Oberfläche ausgeführt werden könnten. Anschließend erfolgt die Zuweisung von Ressourcen wie Arbeitsspeicher und Prozessoranzahl. Für jede neue VM wird dieser Prozess identisch durchgeführt, wodurch sich manuelle Vorgehensweisen mit den PowerShell-basierten Methoden vergleichen lassen. Jedoch wird die Erstellung von mehreren VMs durch ein Skript erheblich vereinfacht, da die Schritte automatisiert und in einer Schleife ausgeführt werden können.

Vor dem Einstieg in weitere Schritte war eine zusätzliche Anpassung hinsichtlich der Netzwerkadapter erforderlich. Da die Systeme über TeamViewer verwaltet werden, musste für den Netzwerkadapter jeder VM eine statische MAC-Adresse vergeben werden. Dies verhindert, dass die Systeme aus der Geräteliste in TeamViewer verschwinden. Um die MAC-Adresse zuweisen zu können, wird die VM einmalig gestartet, sodass Windows eine temporäre Adresse generiert. Anschließend wird die Maschine heruntergefahren und in Hyper-V die MAC-Adresse auf statisch gesetzt. Danach kann der reguläre Arbeitsablauf fortgesetzt werden.

5.3 Verzeichnisstruktur und Logdateien

Zur Verwaltung der Kunden-VMs wurde eine einheitliche Ordnerstruktur implementiert. Hierfür wurde ein eigenes Skript (FilehandlingFunctions) entwickelt, das für jeden Kunden automatisch identische Verzeichnisse anlegt, wobei die Struktur jeweils unter dem individuellen Kundennamen eingetragen wird. Innerhalb dieser Hierarchie werden die vorbereiteten VHD(X)-Dateien aus dem zentralen Vorbereitungsverzeichnis in die entsprechenden Kundenordner kopiert. Vor der Erstellung überprüft das Skript, ob die betreffende Struktur bereits vorhanden ist, um redundante Duplikate zu vermeiden. Die Umsetzung basiert auf einfachem File Handling, wobei neue Verzeichnisse mit dem PowerShell-Befehl $New{-}Item$ erzeugt werden.

Neben der Verzeichnisstruktur wird für jeden Kunden zusätzlich ein Logfile eingerichtet. Dieses ermöglicht sowohl die Nachverfolgung des Skriptablaufs als auch die Identifikation möglicher Fehlerquellen oder Prozessunterbrechungen. Da das VHD(X)-Template für die VM-Erstellung standardmäßig die Datei unattend.xml enthält, in der Administratorpasswörter hinterlegt sind, entsteht ein nicht zu vernachlässigendes Sicherheitsrisiko. Um dies zu vermeiden, wird die unattend.xml nach erfolgreicher Erstellung der Kundenordner sowie der zugehörigen VM mithilfe einer im Skript implementierten Funktion wieder gelöscht.

Für die erfolgreiche Automatisierung war es erforderlich, dass bestimmte XML- und Konfigurationsdateien auf den Zielsystemen verfügbar sind. Um dies zu ermöglichen, musste die Gastdienstschnittstelle innerhalb der VMs aktiviert werden, wodurch sich Dateien direkt vom Hostsystem in die Gastsysteme übertragen lassen. Vor dem eigentlichen Transfer wurde in jeder VM ein Ordner angelegt, in den die relevanten Dateien kopiert werden. Für den Domain Controller waren dies Gruppenrichtlinien-Dateien, während für den Fileserver die Installations-XML der entsprechenden Rolle vorgesehen war.

5.4 Konfiguration der virtuellen Maschinen

Vor der Installation von Rollen ist es notwendig, den VMs statische IP-Adressen zuzuweisen und sowohl dem Fileserver als auch dem Terminalserver den Domain Controller als DNS-Server einzutragen. Darüber hinaus erfolgt eine Anpassung der Computernamen zur eindeutigen Identifikation. Für diesen Zweck wurden spezifische kleine Skripte (ChangeIpRenameTs bzw. ChangeIpRenameDc) entwickelt, die die Konfiguration automatisieren.

Die Installation der Fileserver-Rolle erfolgt gemeinsam mit der Konfiguration einer statischen IP-Adresse, da diese Rolle unabhängig vom Domain Controller eingerichtet wird (DeployFileServerRole).

Die Einrichtung des Domain Controllers basiert ebenfalls auf PowerShell. Bei einer manuellen Installation lässt sich vor der Heraufstufung ein Skript generieren, das die durchgeführten Arbeitsschritte abbildet. Dieses Vorgehen bietet wertvolle Einblicke in die notwendigen Befehle, zusätzlich stehen umfassende Informationen in der offiziellen Microsoft-Dokumentation zur Verfügung. Für die automatisierte Konfiguration wurde eine Hashmap erstellt, die alle erforderlichen Parameter wie Domänenname, NetBIOS-Name, DSRM-Passwort sowie Pfade für System- und Logdateien enthält. Anschließend wurde die benötigte Rolle installiert und der Domain Controller mittels *Install−ADDSForest* heraufgestuft. Nach einem Neustart stand damit die Grundkonfiguration zur Verfügung. Vor dem Neustart wurde zudem der Google-DNS-Server (8.8.8.8) in die Liste der Weiterleitungen aufgenommen.

Zum betrachteten Zeitpunkt standen bereits drei VMs zur Verfügung, von denen zwei mit den vorgesehenen Rollen ausgestattet waren und ein funktionsfähiger Domain Controller bereitgestellt war. Der darauffolgende Schritt bestand darin, die verbleibenden Server in die Domäne aufzunehmen, nachdem der Domain Controller vollständig gestartet war. Dies konnte mit einer dedizierten PowerShell-Funktion erreicht werden, die unter Verwendung von *Invoke−Command* den Domänenbeitritt in den VMs remote ausführt.

Nach erfolgreichem Domänenbeitritt der Systeme folgte die Vorbereitung des Fileservers. Hierzu wurde die zusätzliche Festplatte zunächst online geschaltet und mit dem Partitionsstil GPT initialisiert. Anschließend wurde die gesamte Speicherkapazität in einer Partition zusammengefasst, die mit einem Laufwerksbuchstaben (im Skript: D:) und einer eindeutigen Bezeichnung („Daten“) versehen wurde. In diesem Verzeichnis wurden standardisierte Basisordner angelegt, die später als Grundlage für Freigaben dienen. Da Freigaben mit bestimmten Active-Directory-Gruppen verknüpft sind, war zuvor eine grundlegende AD-Struktur zu etablieren.

Die erstellte Active-Directory-Struktur umfasste exemplarische Benutzerkonten (einen Testbenutzer und einen Administrationsaccount), mehrere Standardgruppen (Scan_ LW, Datevuser, Daten_ LW, GF für Geschäftsführung) sowie Organisationseinheiten zur strukturierten Trennung von Benutzern, Gruppen und Computern. Die Benutzer wurden in Form von Hashmaps angelegt, während Gruppen über den Befehl *New−ADGroup* erstellt wurden.

Im Anschluss wurden die Benutzer den entsprechenden Gruppen zugeordnet.

Zur Grundstruktur des Active Directory gehören auch Standardrichtlinien, etwa für Netzlaufwerke, Remotedesktop-Einstellungen, Einschränkungen der Eingabeaufforderung und Skriptausführung sowie das Deaktivieren von „New Outlook“. Da viele dieser Gruppenrichtlinien auf Registry-Einträgen basieren, können sie mithilfe von Hashmaps umgesetzt werden. Die benötigten Registry-Keys lassen sich entweder durch das Auslesen bestehender Richtlinien über *Get−GPPrefRegistryValue* oder durch Dokumentationen im Internet identifizieren.

Komplexer gestaltet sich die Abbildung von Netzlaufwerken. Die manuelle Konfiguration eines Netzlaufwerks führt zu einer Vielzahl an Registry-Einträgen. Werden diese als Grundlage für eine neue Gruppenrichtlinie übernommen, schlägt die Umsetzung häufig fehl, sodass die Laufwerke nicht im Explorer angezeigt werden. Dadurch werden sie für Endnutzer unbrauchbar. Um dieses Problem zu umgehen, wurde eine bereits funktionierende Gruppenrichtlinie mit den Netzlaufwerken aus einem bestehenden System exportiert, in die VM des Domain Controllers übertragen, an die SID der neuen Domäne angepasst und anschließend importiert.

Die importierten Gruppenrichtlinien (GPOs) verhalten sich nach der Anpassung der SIDs wie manuell erstellte Richtlinien. Ein direkter Eingriff in einzelne Registry-Einträge ist damit nicht erforderlich, da die XML-Dateien der GPOs in einer Schleife verarbeitet und die enthaltenen Gruppen angepasst werden können. Mit Abschluss dieser Schritte stand das Grundgerüst des Active Directory bereit, womit lediglich noch wenige Arbeitsschritte bis zur Fertigstellung des Automatisierungsskripts zur Serverstruktur erforderlich waren.

Um den Zugriff auf die Verzeichnisse des Fileservers zu ermöglichen, wurden die Ordner zunächst freigegeben und mit den notwendigen Berechtigungen versehen. Die Freigabe konnte mit dem Befehl *New−SmbShare* umgesetzt werden, wobei mithilfe von Hashmaps und Schleifen Freigabename, Pfad und Berechtigungen automatisiert zugewiesen wurden. Im Anschluss wurden die NTFS-Berechtigungen auf Verzeichnis- und Dateiebene konfiguriert. Hierfür wurden die bestehenden Rechte eines Verzeichnisses zunächst mit *Get−Acl* in einer Variablen gespeichert, anschließend über eine Hashmap neue Einträge definiert (Benutzer bzw. Gruppe, Berechtigungsumfang sowie Anwendungsbereich: Ordner, Unterordner und Dateien). Diese wurden in ein neues ACL-Objekt überführt und mit *Set−Acl* auf das jeweilige Verzeichnis

angewendet.

Ein Problem dieser Vorgehensweise besteht darin, dass Rechte jeweils nur für einen Ordner und eine Gruppe gleichzeitig gesetzt werden können. Der Versuch, mehrere Einträge parallel zu übernehmen, führte entweder zu fehlerhaften Berechtigungen oder zu fehlenden Fehlermeldungen, sodass die Rechtevergabe stets einzeln vorgenommen werden musste. Nach der Umsetzung der Freigaben und NTFS-Berechtigungen verblieb als letzter Bestandteil die Einrichtung des Terminalservers.

Die Installation der dazugehörigen Rollen konnte nicht vollständig innerhalb des Gastesystems über PowerShell-Befehle erfolgen. Stattdessen wurden die Rollen mithilfe von PowerShell Direct über den Hyper-V-Host remote in die VM installiert und anschließend konfiguriert. Damit konnte eine funktionsfähige Terminalserver-Umgebung bereitgestellt werden, wenngleich die Lizenzierungskonfiguration nicht Teil des Skripts war. Auch ohne diese Konfiguration ist eine Anmeldung mittels Remotedesktop für einzelne Benutzer möglich, wodurch Kernfunktionen des Terminalservers zur Verfügung stehen.

Zum Abschluss wurden sämtliche VMs einmalig neu gestartet, um einen einheitlichen Betriebszustand herzustellen und alle Konfigurationen sowie Änderungen, die einen Neustart erforderten, gültig zu machen. Lediglich beim Terminalserver besteht eine geringe Wahrscheinlichkeit, dass der Dienst "Remotedesktopverwaltung"nach dem Neustart nicht ordnungsgemäßausgeführt wird. Da im Anschluss an die Grundkonfiguration jedoch weitere Installationen und Anpassungen erfolgen, wird dieses Restrisiko als vernachlässigbar eingestuft.

Das Skript verfügt darüber hinaus über ein grafisches Benutzerinterface (GUI), welches die zu Beginn der Einrichtung erforderlichen Eingaben für die Erstellung der virtuellen Maschinen zentral abfragt. Dadurch werden die notwendigen Informationen bereits im Vorfeld gesammelt und automatisch in den weiteren Prozess eingebunden. Dies führt zu einer deutlichen Zeitersparnis im Vergleich zur manuellen Erstellung, da Verzögerungen durch unbeaufsichtigte Eingabefenster, beispielsweise wenn parallel andere Tätigkeiten ausgeführt werden, vermieden werden können.

6 Testumgebungen

Zur Validierung des entwickelten Skripts wurden Testdurchläufe in zwei unterschiedlichen Umgebungen durchgeführt. Dadurch kann die Stabilität und Portabilität der Lösung besser eingeschätzt werden, da sowohl die eingesetzte Hardware als auch die Software variieren.

Hardware- und Softwarekonfiguration

Tabelle 1: Hardware- und Softwarekonfiguration der Testumgebungen

Komponente	**Testumgebung Home**	**Testumgebung Server 1**	**Testumgebung Server 2**
CPU	AMD Ryzen 7 7800X3D, 8 Kerne / 16 Threads, 4,3–4,5 GHz	Intel Xeon E3-1230 v5, 4 Kerne / 8 Threads, 3,7 GHz	2× Intel Xeon E5-2643 v3, 6 Kerne / 12 Threads
RAM	32 GB DDR5-6000	64 GB DDR4-2133	256 GB DDR4-2133
PowerShell-Version	7.5.2	5.1	5.1
Windows-Version	Windows 11 Pro	Windows Server 2025 Standard	Windows Server 2019 Standard

Virtuelle Maschinen

Die virtuellen Maschinen wurden in beiden Testumgebungen identisch konfiguriert:

- **CPU-Kerne:** 2
- **RAM:** 2GB pro VM
- **Zusätzliche VHDX:** 35MB auf dem File Server

Testmethodik und Testergebnisse

Zur Überprüfung des entwickelten Skripts wurden mehrere Testdurchläufe auf den beschriebenen Umgebungen durchgeführt. Ziel war die Validierung der Funktionsfähigkeit, Stabilität und Ausführungsdauer des Automatisierungsprozesses. Der Fokus lag dabei nicht auf einer detaillierten funktionalen Prüfung der konfigurierten Dienste, sondern auf der wiederholbaren und automatisierten Erstellung der virtuellen Maschinen. Trotzdem wurde darauf geachtet, dass die grundlegenden Konfigurationen wie Domänenbeitritt,

Netzwerkanpassungen und Freigaben korrekt umgesetzt wurden und funktionsfähig sind.

Tabelle 2: Zeitübersicht der Testdurchläufe – Server 1

Schritt	**Log 1 (hh:mm:ss)**	**Log 2 (hh:mm:ss)**	**Log 3 (hh:mm:ss)**	**Log 4 (hh:mm:ss)**	**Log 5 (hh:mm:ss)**
Skript-Start	0:00:00	0:00:00	0:00:00	0:00:00	0:00:00
Kopieren fertig	0:00:26	0:00:31	0:00:32	0:00:31	0:00:32
VMs erstellt	0:00:30	0:00:36	0:00:36	0:00:34	0:00:36
VMs angepasst	0:00:31	0:00:36	0:00:36	0:00:35	0:00:37
VMs gestartet	0:00:34	0:00:39	0:00:39	0:00:37	0:00:40
Windows initialisierung fertig	0:01:13	0:01:34	0:01:30	0:01:28	0:01:32
Kopieren fertig	0:01:44	0:02:04	0:01:56	0:02:01	0:02:04
Löschen von Dateien fertig	0:01:52	0:02:14	0:02:08	0:02:10	0:02:14
MAC umgestellt	0:02:05	0:02:36	0:02:21	0:02:23	0:02:24
static IP	0:02:32	0:03:03	0:02:46	0:02:49	0:02:50
DC installiert	0:04:21	0:04:52	0:05:08	0:04:48	0:05:16
Domain beigetreten	0:10:01	0:11:32	0:11:48	0:11:28	0:11:56
Ordnerstruktur FS	0:11:56	0:12:57	0:12:43	0:12:21	0:12:52
AD Struktur erstellt	0:12:19	0:13:20	0:13:06	0:12:44	0:13:15
Freigaben Ordner	0:12:25	0:13:24	0:13:13	0:12:51	0:13:24
TS installiert	0:18:13	0:18:14	0:19:09	0:18:32	0:19:04
Skript-Ende	0:18:40	0:18:43	0:19:10	0:18:33	0:19:04
Gesamtzeit	0:18:40	0:18:43	0:19:10	0:18:33	0:19:04

Tabelle 3: Zeitübersicht der Testdurchläufe – Server 1

Schritt	**Log 1 (hh:mm:ss)**	**Log 2 (hh:mm:ss)**	**Log 3 (hh:mm:ss)**	**Log 4 (hh:mm:ss)**	**Log 5 (hh:mm:ss)**
Skript-Start	0:00:00	0:00:00	0:00:00	0:00:00	0:00:00
Kopieren fertig	0:06:02	0:06:03	0:04:53	0:04:53	0:04:52
VMs erstellt	0:06:15	0:06:14	0:05:03	0:05:04	0:05:04
VMs angepasst	0:06:15	0:06:14	0:05:04	0:05:04	0:05:04
VMs gestartet	0:06:20	0:06:19	0:05:09	0:05:09	0:05:09
Windows initialisierung fertig	0:10:52	0:12:12	0:09:52	0:11:17	0:10:21
Kopieren fertig	0:12:30	0:13:33	0:11:28	0:12:37	0:12:14
Löschen von Dateien fertig	0:13:00	0:13:51	0:11:45	0:13:58	0:12:34
MAC umgestellt	0:14:34	0:15:31	0:13:23	0:14:06	0:13:39
static IP	0:16:30	0:17:28	0:15:14	0:16:07	0:15:28
DC installiert	0:21:01	0:22:47	0:19:56	0:20:50	0:20:33
Domain beigetreten	0:27:52	0:29:31	0:26:38	0:27:37	0:27:26
Ordnerstruktur FS	0:29:12	0:30:55	0:28:03	0:29:12	0:29:20
AD Struktur erstellt	0:29:53	0:31:41	0:28:42	0:30:10	0:30:06
Freigaben Ordner	0:30:03	0:31:51	0:28:52	0:30:22	0:30:17
TS installiert	0:42:19	0:43:29	0:40:47	0:43:08	0:41:37
Skript-Ende	0:43:11	0:43:29	1:09:41	0:43:08	0:41:37
Gesamtzeit	0:43:11	0:43:29	1:09:41	0:43:08	0:41:37

Tabelle 4: Zeitübersicht der Testdurchläufe – Server 2

Schritt	**Log 1 (hh:mm:ss)**	**Log 2 (hh:mm:ss)**	**Log 3 (hh:mm:ss)**	**Log 4 (hh:mm:ss)**	**Log 5 (hh:mm:ss)**
Skript-Start	0:00:00	0:00:00	0:00:00	0:00:00	0:00:00
Kopieren fertig	0:00:30	0:00:30	0:00:30	0:01:53	0:00:29
VMs erstellt	0:03:05	0:02:35	0:03:46	0:05:58	0:03:25
VMs angepasst	0:03:05	0:02:35	0:03:46	0:05:58	0:03:25
VMs gestartet	0:03:12	0:02:44	0:03:54	0:06:07	0:03:34
Windows initialisierung fertig	0:04:19	0:04:19	0:05:31	0:06:51	0:05:02
Kopieren fertig	0:05:10	0:05:13	0:06:31	0:07:41	0:06:07
Löschen von Dateien fertig	0:05:52	0:05:41	0:07:13	0:08:10	0:06:49
MAC umgestellt	0:06:12	0:06:22	0:07:56	0:08:58	0:07:35
static IP	0:06:51	0:07:08	0:08:40	0:09:44	0:08:28
DC installiert	0:10:50	0:10:31	0:12:15	0:13:16	0:11:50
Domain beigetreten	0:17:38	0:17:16	0:19:37	0:20:02	0:18:36
Ordnerstruktur FS	0:18:22	0:17:56	0:20:03	0:21:07	0:19:42
AD Struktur erstellt	0:18:32	0:18:30	0:20:43	0:21:44	0:19:50
Freigaben Ordner	0:18:58	0:18:40	0:20:53	0:21:54	0:20:00
TS installiert	0:29:00	0:29:02	0:31:24	0:31:53	0:31:02
Skript-Ende	0:32:01	0:29:31	0:32:35	0:54:02	0:31:29
Gesamtzeit	0:32:01	0:29:31	0:32:35	0:54:02	0:31:29

Die Messungen der Skriptausführungszeiten in den drei unterschiedlichen Testumgebungen zeigen deutliche Unterschiede in der Performance, die auf die jeweiligen Hardware- und Netzwerkkonfigurationen zurückzuführen sein dürften. In der häuslichen Testumgebung lagen die Gesamtdauern der Skriptdurchläufe zwischen 18:33 und 19:10 Minuten (siehe Table 2). Die einzelnen Schritte, wie das Erstellen und Anpassen der VMs oder die Initialisierung von Windows, dauerten hier nur wenige Sekunden bis wenige Minuten, was auf eine schnelle lokale Verarbeitung und geringe Latenzen hinweist.
Im Vergleich dazu zeigten die Tests auf Server 1 signifikant längere Ausführungszeiten. Die Gesamtdauer bewegte sich zwischen 41:37 und 1:09:41 Stunden (siehe Table 3), was insbesondere bei den Schritten „Domain beigetreten“ und „TS installiert“ deutlich sichtbar wird. Die erhöhte Dauer lässt sich vermutlich durch Netzwerkverzögerungen, höhere Serverlast oder parallele Prozesse erklären, die die Skriptausführung verlangsamen.
Die Testumgebung auf Server 2 zeigte hingegen eine moderate Ausführungszeit zwischen 29:00 und 31:53 Minuten (siehe Table 4). Die Schritte „VMs erstellt“ und „Windows Initialisierung fertig“ beanspruchten hier vergleichsweise mehr Zeit als in der häuslichen Umgebung, jedoch deutlich weniger als auf Server 1. Dies deutet auf eine Serverkonfiguration hin, die effizienter arbeitet als Server 1, jedoch langsamer ist als die lokale Umgebung.
Zusammenfassend lässt sich festhalten, dass die Ausführungszeiten stark von der Testumgebung abhängen. Die lokale Umgebung zeigt die kürzesten Zeiten, Server 2 liegt im mittleren Bereich, während Server 1 aufgrund von längeren Wartezeiten und möglichen Performance-Engpässen die längsten Zeiten aufweist. Diese Ergebnisse unterstreichen die Bedeutung von Hardware- und Netzwerkfaktoren bei der automatisierten Skriptausführung und sollten bei der Planung und Optimierung von Deployment-Strategien berücksichtigt werden.

Ebenfalls wurde die Einrichtung der VMs auf der Testumgebung mit dem Server 1 manuell getestet. Der erste, manuelle, Test wurde von einer Praktikantin mit begleitender Unterstützung durchgeführt. In etwas 2 Stunden war das System aus 3 VMs genauso einsatzfähig wie die Serverstruktur, die durch das Skript automatisiert erstellt wurde. Drei weitere manuelle Test wurden vom Autor dieser Arbeit durchgeführt. Durch die Erfahrungen, die während der Entwicklung des Skripts gesammelt wurden, konnte die Zeit der manuelle Einrichtung merkbar reduziert werden. Anfangs war die manuelle Einrichtung der VMs langsamer und mit 1 Stunde und 33 Minuten deutlisch länger als die automatisierte Variante. Doch mit der wachsenden Routine, durch die gleichen Abläufe, konnte die Zeit im dritten Durchlauf auf 1 Stunde und 8 Minuten reduziert werden. Das ist jedoch kein Vergleich zu der Automatisierung durch das Skript

7 Technische Erkenntnisse der Skripterstellung

Während der Entwicklung und Umsetzung des Automatisierungsskripts für Hyper-V traten eine Vielzahl technischer Besonderheiten und Fallstricke auf, die bei zukünftigen Projekten berücksichtigt werden sollten. Eine grundlegende Voraussetzung für bestimmte Aktionen, wie beispielsweise den Domänenbeitritt, ist die Verwendung eindeutiger SIDs. Dies wurde durch den Einsatz von *sysprep* in Verbindung mit einer funktionierenden Antwortdatei sichergestellt. Dabei zeigte sich, dass die Antwortdatei strikt in der Reihenfolge von oben nach unten abgearbeitet wird, weshalb der *OOBE*-Abschnitt möglichst früh platziert werden sollte.

Im Bereich der Skripterstellung mit PowerShell war besonders bei der Arbeit mit GUIs und Benutzerinteraktionen Aufmerksamkeit erforderlich. So erwies es sich als sinnvoll, beim Aufbau einer RAM-Auswahl Integer-Werte in Byte-Form anzugeben, da Strings in Arrays nicht zuverlässig verarbeitet wurden. Das Rückgabeverhalten wurde über *return* angepasst, um korrekte Werte zu liefern. Außerdem zeigte sich, dass in PowerShell die Überprüfung auf *NULL* konsistent mit der Schreibweise *NULL −eq \$Variable* erfolgen sollte, um logische Fehler, insbesondere bei Array-Prüfungen,zu vermeiden.

GUI-Fenster verhielten sich in Bezug auf die Anzeige im Vordergrund nicht im-

mer wie erwartet, selbst wenn die Eigenschaft *TopMost* gesetzt war. Dieses Problem wurde durch die Definition eines Mutterfensters und die gezielte Anzeige untergeordneter Fenster mit *.Add_Shown()* gelöst. Passworteingaben ließen sich in der GUI zwar ausblenden, mussten jedoch für die weitere Verarbeitung mittels *ConvertTo−SecureString −PlainText* in ein SecureString-Format konvertiert werden. Beim Auswählen von Verzeichnissen war zu beachten, dass der Pfad über die Eigenschaft *InitialDirectory* und nicht *Root Folder* festgelegt werden muss.

Für die Eingabevalidierung, beispielsweise bei IP-Adressen, bot sich die Nutzung regulärer Ausdrücke wie *"[0−9]+\.[0−9]+\.[0−9]+\.[0−9]+"* an. Variablen, die innerhalb von Remote-Jobs oder Sessions genutzt werden, mussten mit dem Präfix *Using:* übergeben werden – auch dann, wenn sie zuvor global deklariert wurden. Innerhalb von Strings war eine Variablenersetzung nur mit der Schreibweise *\$(Variable)* möglich. Bei Abfragen, die einen String erfordern, erwies sich die Verwendung des Parameters *−ExpandProperty* als notwendig, um Objektrückgaben zu vermeiden.

Das Ausführen von Befehlen mit dem Parameter *−AsJob* startete Prozesse im Hintergrund, was problematisch sein konnte, wenn das Ergebnis sofort benötigt wurde. In solchen Fällen war es erforderlich, den Job aktiv zu überwachen, das Ende der Ausführung abzuwarten und anschließend den Job zu entfernen. Beim Einsatz von *Invoke−Command* konnte entweder direkt mit VM-Namen oder mit zuvor erstellten Sessions gearbeitet werden; letztere mussten nach Abschluss wieder geschlossen werden. Um Anmeldevorgänge zu vermeiden, konnten Anmeldeinformationen als *PSCredential*-Objekte hinterlegt werden. Diese wurden über *New−Object* erstellt, wobei das Passwort als SecureString abgefragt und zusammen mit dem Benutzernamen in einer Variablen gespeichert wurde.

Vor der Ausführung des Skripts wurde geprüft, ob es in einer administrativen Sitzung lief. Falls nicht, wurde es über *Start−Process* mit dem Verb *RunAs* neu gestartet. Für eine einheitliche Protokollierung wurde eine Log-Funktion erstellt, die Zeitstempel im Format *"[dd/MM/yy HH:mm:ss]"* generierte. Nicht benötigte Ausgaben wurden konsequent mit Out-Null| unterdrückt.

Netzwerkanpassungen innerhalb von Windows-VMs erfolgten mit *New−\ allowbreak Net\ allowbreak IPAddress* anstelle von *Set−NetIPAddress*. Um Dateien vom Hyper-V-Host auf eine VM zu übertragen, musste die Gastdienstschnittstelle aktiviert werden, was über *Enable−VMIntegrationService* möglich war. Anschließend konnte der Kopiervorgang mit *Copy−VMFile* durchge-

führt werden, wobei Systempfade nur indirekt beschreibbar waren.

Die Installation bestimmter Serverrollen, wie des Dateiservers, war nur über XML-basierte Konfigurationen möglich. Remote Desktop Services (RDS) konnten vom Domänencontroller auf den Terminalserver installiert werden, wobei zusätzliche Konfigurationen – etwa Session Collections, RDS-Lizenzierung und Neustarts des Verwaltungsdienstes – direkt auf dem TS erfolgten.
Weitere Besonderheiten betrafen die Handhabung von Strings, Hash-Tabellen und VM-Eigenschaften: Beim Splitten eines Strings am Punkt musste der Punkt escaped werden.
VMs konnten vollständig aus Hash-Tabellen heraus erstellt werden, wobei CPU-Anpassungen erst nach der Erstellung und MAC-Adressänderungen nach dem ersten Start möglich waren. Neue virtuelle Festplatten wurden per *Add−VMHardDiskDrive* angebunden, anschließend online geschaltet, benannt, mit einem Laufwerksbuchstaben versehen und formatiert.

Passwörter von Active-Directory-Konten ließen sich nicht ohne Weiteres zurücksetzen, während lokale Passwörter per Remote- oder PowerShell-Direct-Zugriff problemlos geändert werden konnten. Bestimmte Installationsfehlermeldungen bei RDS ließen sich möglicherweise auf fehlende Lizenzserverkonfigurationen zurückführen.

8 Fazit

Das Zield er Arbeit war die Erstellung und Konfiguration von Windows-basierten virtuellen Maschinen in Hyper-V durch den Einsatz von PowerShell zu automatisieren. Ausgangspunkt war die übliche manuelle Vorgehensweise, bei der zahlreiche wiederkehrende Schritte erforderlich sind, um Domänencontroller, Dateiserver und Terminalserver bereitzustellen.

Das entwickelte Powershell-Skript übernimmt die wesentliche Aufgaben der Systemeinrichtung. Dazu zählen die automatisierte Erstellung von VMs auf Basis eines vorbereiteten Windows-Server-Templates, die Vergabe statischer IP-Adressen und Computernamen, die Konfiguration von Active Directory auf einemdem dazu Eingerichteten Domain Controller, die Einrichtung eines Fileserver und die dazugehörigen Freigaben sowie die Zuweisung von NTFS-Berechtigungen. Ergänzend wurde die Konfiguration des Terminalservers einschließlich der benötigten Rollenkomponenten umgesetzt. Durch diese Automatisierung konnte der manuelle Aufwand erheblich reduziert und die Gefahr von Fehlern in der Einrichtung minimiert werden.

Die Tabellen zur Umsetzung verdeutlichen, dass ein hoher Automatisierungsgrad erreicht wurde. Dennoch bleiben einzelne manuelle Nacharbeiten, wie etwa die Aktivierung des Betriebssystems oder die finale Lizenzierung des Terminalservers, notwendig. Dies stellt eine Grenze der aktuellen Lösung dar, die in zukünftigen Weiterentwicklungen durch zusätzliche Skriptmodule oder den Einsatz weiterführender Tools adressiert werden könnte.

Zusammenfassend zeigt die Arbeit, dass der Einsatz von PowerShell ein effizientes Mittel darstellt, um wiederkehrende und fehleranfällige Administrationsaufgaben in Hyper-V-Umgebungen zu automatisieren. Damit leistet sie einen Beitrag zur Standardisierung von Abläufen in der Windows-Server-Administration, insbesondere für kleine und mittlere Unternehmen.

Darüber hinaus ergeben sich aus der Arbeit verschiedene Ansätze zur Weiterentwicklung des Skripts. So wäre es sinnvoll, die Funktionalität dahingehend zu erweitern, dass optional auch nur zwei oder eine virtuelle Maschine erstellt werden kann, um die Lösung besser an kleinere Kundenszenarien anzupassen. Ebenso könnte eine flexible Erweiterung des Speichers einzelner VMs implementiert werden, idealerweise über eine grafische Oberfläche zur vereinfachten Bedienung.

Weitere Verbesserungsmöglichkeiten bestehen insbesondere im Bereich der Terminalserver-Konfiguration, beispielsweise durch die automatische Einbindung des Lizenzservers. Auch die Anpassung des Skripts an Failover-Cluster stellt einen relevanten nächsten Schritt dar, da die aktuelle Entwicklung ausschließlich in einer Standalone-Hyper-V-Umgebung getestet wurde und cluster-spezifische Rollen und Eigenschaften noch nicht berücksichtigt werden.

Ein weiterer Optimierungsbedarf betrifft die derzeit implementierte Wartezeit nach der Installation des Domänencontrollers. Momentan wird hierfür ein statisches Zeitintervall von 6,5 Minuten per sleep-Befehl genutzt. Dieses Vorgehen ist zwar funktional, jedoch unflexibel und wenig professionell. Eine zuverlässigere Methode zur Erkennung des erfolgreichen Systemstarts wäre hier wünschenswert, auch wenn die Beobachtung und zeitliche Abschätzung des Bootvorgangs von Windows-Servern eine gewisse Herausforderung darstellt.

Schließlich könnte die Benutzerverwaltung verbessert werden, indem Benutzerkonten und Gruppen über eine XML-Datei definiert und direkt nach der Erstellung automatisch verarbeitet werden. Auf diese Weise ließe sich der Grad der Standardisierung und Automatisierung nochmals erhöhen.

Erklärung zum Einsatz von KI-Werkzeugen

Während der Erstellung dieser Bachelorarbeit wurden KI-gestützte Hilfsmittel zur Optimierung des sprachlichen Ausdrucks und der Textstruktur verwendet. Der Einsatz beschränkte sich ausschließlich auf Aspekte der Lesbarkeit und Formulierung. Eine eigenständige inhaltliche Generierung oder Erweiterung der wissenschaftlichen Ergebnisse durch KI erfolgte nicht.
Die folgenden Prompts wurden im Zuge der Bearbeitung eingesetzt:

Perplexity AI

> Act like a professional academic editor and scientific writing assistant. You specialize in transforming raw, unstructured descriptive text into a polished, logically structured, and academically styled document suitable for a bachelor's thesis, research report, or academic article.
>
> Objective: Your task is to take unstructured input text (such as everyday experiences, descriptive reports, or freeform notes) and convert it into a refined, scientific-style output text. The resulting text must be logically structured, written in a formal academic tone, and formatted consistently. The original meaning and content must remain unchanged, but the phrasing, clarity, and structure should be optimized.
>
> Instructions: 1. Input format: You will receive an unstructured, descriptive text (e.g., an everyday report or narrative). 2. Output format: Provide the revised text in one of the following styles: - LaTeX syntax with \section, \subsection, and \paragraph blocks for clear structuring - OR a polished academic continuous text formatted like a bachelor's thesis or scientific report. Choose the style most appropriate for the input text unless explicitly instructed otherwise. 3. Writing style requirements: - Use formal academic language. - Ensure clarity, conciseness, and logical flow. - Apply a passive writing style (third-person perspective). - Remove redundancies and improve readability. - Correct grammar, spelling, and syntax errors. 4. Structure guidelines: - Introduce a clear title or heading structure. - Divide the text into logical sections and subsections (e.g., Introduction, Methods, Results, Discussion, Conclusion) when suitable. - Ensure coherent transitions between sections. 5. Content requirements: - Do not alter the factual meaning of the text. - Improve the logical order and

argumentative flow. - Highlight cause-effect relationships and analytical depth where appropriate. 6. Output length: The revised academic text should be at least as long as the input, ideally more detailed and better structured. 7. Example structure (if LaTeX is used): \section{Introduction} \subsection{Background} \subsection{Objective} \section{Methods} \section{Results} \section{Discussion} \section{Conclusion}

Final Step: Always deliver the rewritten output in the chosen academic structure without adding commentary or meta-explanations.

Take a deep breath and work on this problem step-by-step.

ChatGPT (GPT-5)

Kannst du mir aus diesen Angaben einen Promt erstellen?

Was die Eingabe ist: unstrukturierter beschreibender Text, evtl. Alltags-/Erfahrungsbericht aus der Ich-Perspektive.

Was die Ausgabe sein soll: ein wissenschaftlich strukturierter, sprachlich verbesserter Text mit Kapiteln/Abschnitten.

Wie die Form sein soll: z. B. LaTeX-Syntax mit \section-Blöcken oder ein formatierter Fließtext für eine Bachelorarbeit o. Ä.

Worauf beim Umschreiben geachtet werden soll: keine inhaltliche Veränderung, aber sprachliche Straffung, logischere Struktur, passiver Schreibstil (3. Person).

Literatur

[1] Hasan Fayyad-Kazan, Luc Perneel, and Martin Timmerman. Benchmarking the performance of microsoft hyper-v server, vmware esxi and xen hypervisors. *Journal of Emerging Trends in Computing and Information Sciences*, 4(12):922–933, 2013.

[2] Yutaka Haga, Kazuhide Imaeda, and Masayuki Jibu. Windows server 2008 r2 hyper-v server virtualization. *Fujitsu Sci. Tech. J*, 47(3):349–355, 2011.

[3] Meshal Fawaz Aldhamen. Windows server management.

[4] L Edward Garren. *Utilizing Memory Analysis of Windows 10 Systems to Identify Malicious Attacks via PowerShell*. PhD thesis, Capitol Technology University, 2025.

[5] Microsoft. Verwalten virtueller windows-computer mit powershell direct. `https://learn.microsoft.com/de-de/windows-server/virtualization/hyper-v/powershell-direct`, 2025. Accessed: 2025-08-21.

[6] Microsoft. Sysprep (generalize) a windows installation. `https://learn.microsoft.com/en-us/windows-hardware/manufacture/desktop/sysprep--generalize--a-windows-installation?view=windows-11`, 2021. Accessed: 2025-08-21.

[7] Microsoft. Sysprep (system preparation) overview. `https://learn.microsoft.com/en-us/windows-hardware/manufacture/desktop/sysprep--system-preparation--overview?view=windows-11`, 2021. Accessed: 2025-08-21.

[8] Microsoft. Microsoft.powershell.core module (powershell). `https://learn.microsoft.com/de-de/powershell/module/microsoft.powershell.core/?view=powershell-7.5`, 2025. Accessed: 2025-08-21.

[9] Microsoft. Microsoft.powershell.management module (powershell). `https://learn.microsoft.com/de-de/powershell/module/microsoft.powershell.management/?view=powershell-7.5`, 2025. Accessed: 2025-08-21.

[10] Microsoft. Microsoft.powershell.security module (powershell). https://learn.microsoft.com/de-de/powershell/module/microsoft.powershell.security/?view=powershell-7.5, 2025. Accessed: 2025-08-21.

[11] Microsoft. Microsoft.powershell.utility module (powershell). https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.utility/?view=powershell-7.5, 2025. Accessed: 2025-08-21.

[12] Microsoft. Nettcpip module (powershell). https://learn.microsoft.com/en-us/powershell/module/nettcpip/?view=windowsserver2025-ps, 2025. Accessed: 2025-08-21.

[13] Microsoft. Netadapter module (powershell). https://learn.microsoft.com/en-us/powershell/module/netadapter/?view=windowsserver2025-ps, 2025. Accessed: 2025-08-21.

[14] Microsoft. Dnsserver module (powershell). https://learn.microsoft.com/en-us/powershell/module/dnsserver/?view=windowsserver2025-ps, 2025. Accessed: 2025-08-21.

[15] Microsoft. Dnsclient module (powershell). https://learn.microsoft.com/en-us/powershell/module/dnsclient/?view=windowsserver2025-ps, 2025. Accessed: 2025-08-21.

[16] Microsoft. Activedirectory module (powershell). https://learn.microsoft.com/en-us/powershell/module/activedirectory/?view=windowsserver2025-ps, 2025. Accessed: 2025-08-21.

[17] Microsoft. Grouppolicy module (powershell). https://learn.microsoft.com/en-us/powershell/module/grouppolicy/?view=windowsserver2025-ps, 2025. Accessed: 2025-08-21.

[18] Microsoft. Storage module (powershell). https://learn.microsoft.com/en-us/powershell/module/storage/?view=windowsserver2025-ps, 2025. Accessed: 2025-08-21.

[19] Microsoft. Servermanager module (powershell). https://learn.microsoft.com/en-us/powershell/module/servermanager/?view=windowsserver2025-ps, 2025. Accessed: 2025-08-21.

[20] Microsoft. Hyper-v module (powershell). https://learn.microsoft.com/en-us/powershell/module/hyper-v/?view=windowsserver2025-ps, 2025. Accessed: 2025-08-21.

[21] Microsoft. Remotedesktop module (powershell). https://learn.microsoft.com/en-us/powershell/module/remotedesktop/?view=windowsserver2025-ps, 2025. Accessed: 2025-08-21.

[22] Microsoft. Creating a custom input box (powershell sample). https://learn.microsoft.com/en-us/powershell/scripting/samples/creating-a-custom-input-box?view=powershell-7.5, 2025. Accessed: 2025-08-21.