ScriptRunner®

FACHARTIKEL

Automatisierung mit der PowerShell

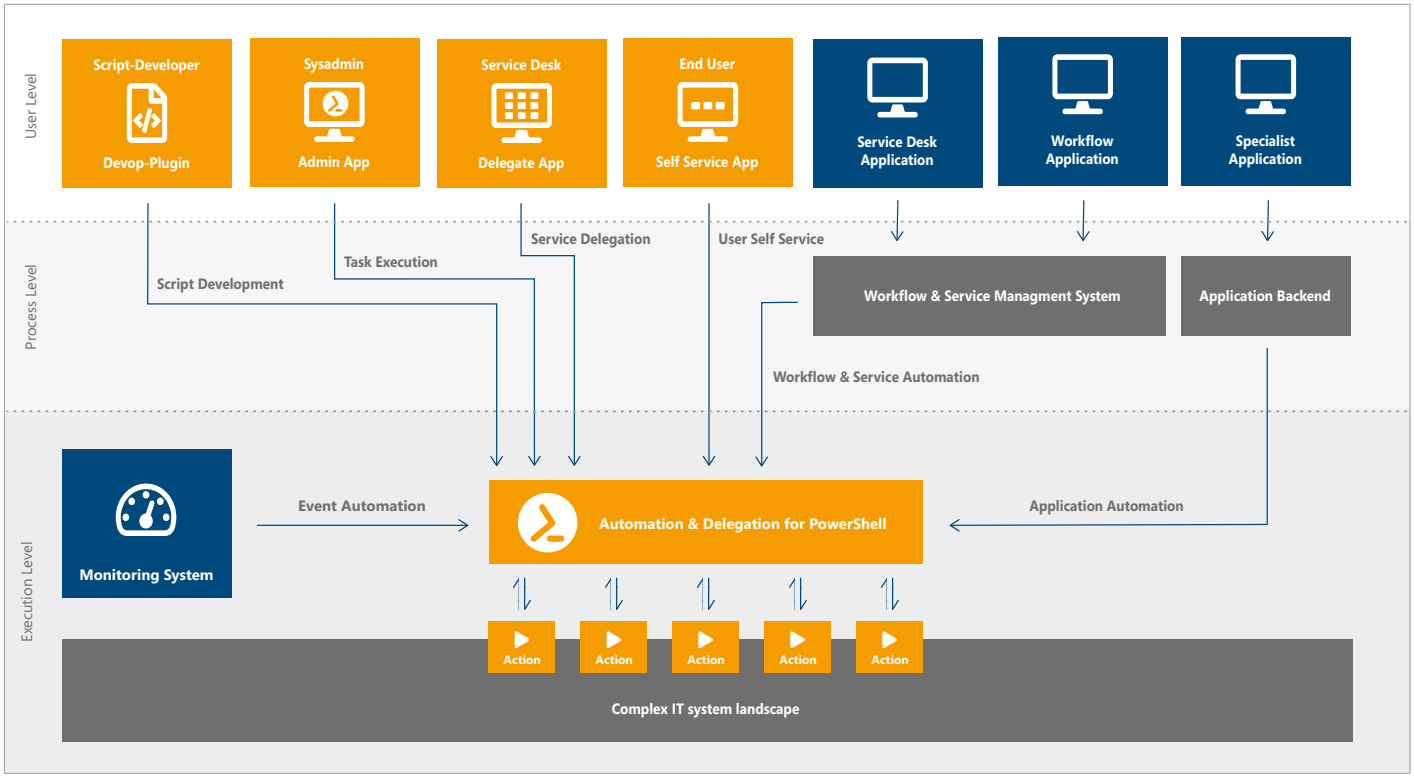
VON FRANK KRESSE

Administratoren wenden einen großen Teil ihrer Arbeit für stets wiederkehrende Anwendungsfälle auf. Durch die Automatisierung von Aufgaben und eine Delegation von Tätigkeiten lässt sich viel Zeit sparen. Der Artikel schildert Herausforderungen und Erfahrungen beim Umgang mit der PowerShell und erklärt anhand eines Beispiels, wie Administratoren vom Cmdlet über das Script zu einer Vorgangsautomation gelangen. Dabei gehen wir auch auf eine mögliche Sicherheitsarchitektur für die PowerShell in automatisierten IT-Prozessen ein.

Ein wesentlicher Bestandteil jeder Digitalisierungsstrategie ist die Automation von wiederkehrenden Aufgaben, Routinevorgängen und ganzen Geschäftsprozessen bis hin zur automatischen Reaktion auf sich wiederholende Ereignisse. Damit greift die IT tiefer als jemals zuvor in das Kerngeschäft eines jeden Unternehmens ein. Auch der IT-Betrieb selbst steht im Fokus von Digitalisierung und Automatisierung. Trotzdem bleibt der Automatisierungsgrad oft weit hinter den Möglichkeiten zurück.

Bei einem genaueren Blick auf die Automation in verschiedenen Bereichen der IT-Infrastruktur ergibt sich ein gemischtes Bild: In der Vergangenheit lag ein starker Fokus auf Automatisierungslösungen für Softwareverteilung. Dieser Teilbereich ist in vielen Unternehmen inzwischen etabliert. Die zweite, aktuell noch laufende Welle betrifft die Automatisierung der Systemüberwachung mit Monitoring-Systemen. Ein Nachteil bleibt dabei jedoch: Es muss immer noch manuell auf die gemeldeten Events reagiert werden. Was das IT-Service Management angeht, sind Ticketsysteme weitgehend etabliert. In der Arena der Service Automation, inklusive User Self Services, gibt es bei den Unternehmen jedoch unterschiedlich große weiße Flecken auf der Automatisierungslandkarte.

ScriptRunner®



ZWEI STRATEGIEN ZUR AUTOMATION

Für die Gestaltung einer Automationsstrategie bieten sich zwei grundverschiedene Ansätze an: **Top-Down oder Bottom-Up.** Beim Top-Down-Ansatz stehen Lösungen und Produkte für die Vollautomation der IT-Prozesse im Fokus. Dieser Ansatz wird oftmals sehr visionär getrieben und lässt operative Erfordernisse und Rahmenbedingungen häufig außen vor. In der Praxis führt das nicht selten zu sehr großen, komplexen und ausufernden Projekten. Diese adressieren oft fortgeschrittene Stufen einer Transformation, ohne dabei die Eingangsstufen bereits bewältigt zu haben.

Typisch für den Top-Down-Ansatz sind:

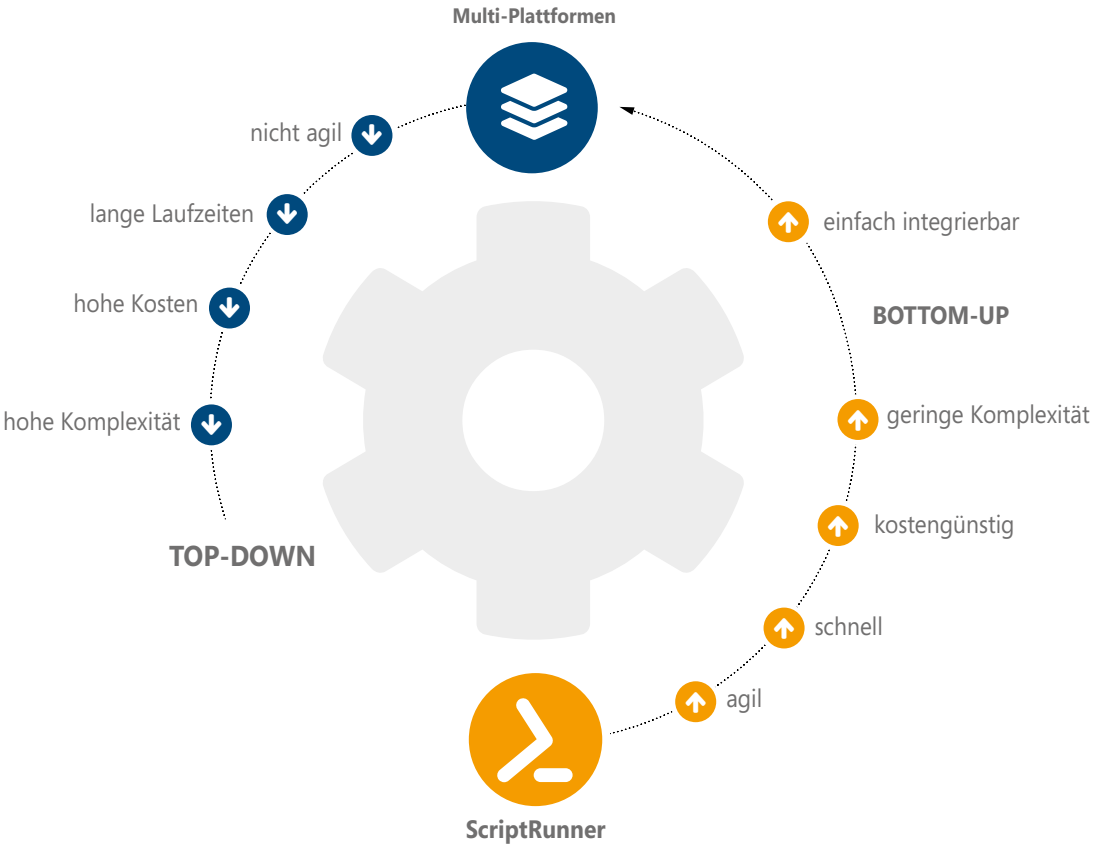
- wenige und sehr späte Ergebnisse für die tatsächlichen operativen Belange aus dem Tagesbetrieb
- sehr hohe Kosten und lange Laufzeiten, unter anderem wegen der enormen Komplexität und des massiven Umfangs
- fehlende Agilität bei oftmals schlechter Kosten-Nutzen-Relation
- zusätzliche hohe Arbeitsbelastung der IT-Teams sowie viele erforderliche neue Skills und gleichzeitig viel Spezialisten-Know-how
- hohes Risikopotenzial bei einer Zielverfehlung oder dem Scheitern des Projekts

Der Bottom-Up-Ansatz hingegen geht ausschließlich von den operativen Erfordernissen und Brennpunkten im IT-Betrieb aus. Er stellt schnelle Verbesserungen mit einem agilen Vorgehen in den Fokus. Mit Bottom-Up-Strategien entscheiden sich Unternehmen für ein pragmatisches Vorgehen, einfachere Plattformen und übersichtliche Projekte.

Daher zeigt sich beim Bottom-Up-Ansatz meist folgendes Bild:

- schnelle Ergebnisse im Tagesbetrieb
- verteilte und geringere Kosten, deutlich reduzierte Komplexität und ein überschaubarer Arbeitsumfang in den einzelnen Stufen der Automatisierung
- hohe Agilität bei sehr guter Kosten-Nutzen-Relation
- kurze Projektlaufzeiten mit jeweils wenigen Abhängigkeiten
- überschaubare zusätzliche Arbeitsbelastung und schrittweiser Skill- und Erfahrungsaufbau im Team
- besser beherrschbares Risiko auf dem Weg zur vollautomatisierten IT-Infrastruktur

ScriptRunner®



Beide Strategien führen letztlich zu einer Gesamt-Automationslösung, allerdings auf verschiedenen Wegen und mit sehr unterschiedlichen Zwischenergebnissen. Während der Bottom-Up-Ansatz schnelle Ergebnisse für tägliche Aufgabenbewältigung liefert, versucht der Top-Down-Ansatz zuerst eine umfassende Lösung für die komplette Prozessebene zu definieren. Der wesentliche Vorteil der Bottom-Up-Strategie ist es, die Steuerungs- und Prozessebene schrittweise zu erschließen, nachdem die Einzelaufgaben bereits erfolgreich automatisiert wurden. Aus Sicht eines agilen Zyklus aus Konzipieren, Implementieren, Lernen und Verbessern ist dies deutlich leichter umzusetzen.

DIE POWERSHELL ALS GRUNDLAGE FÜR IT-AUTOMATION

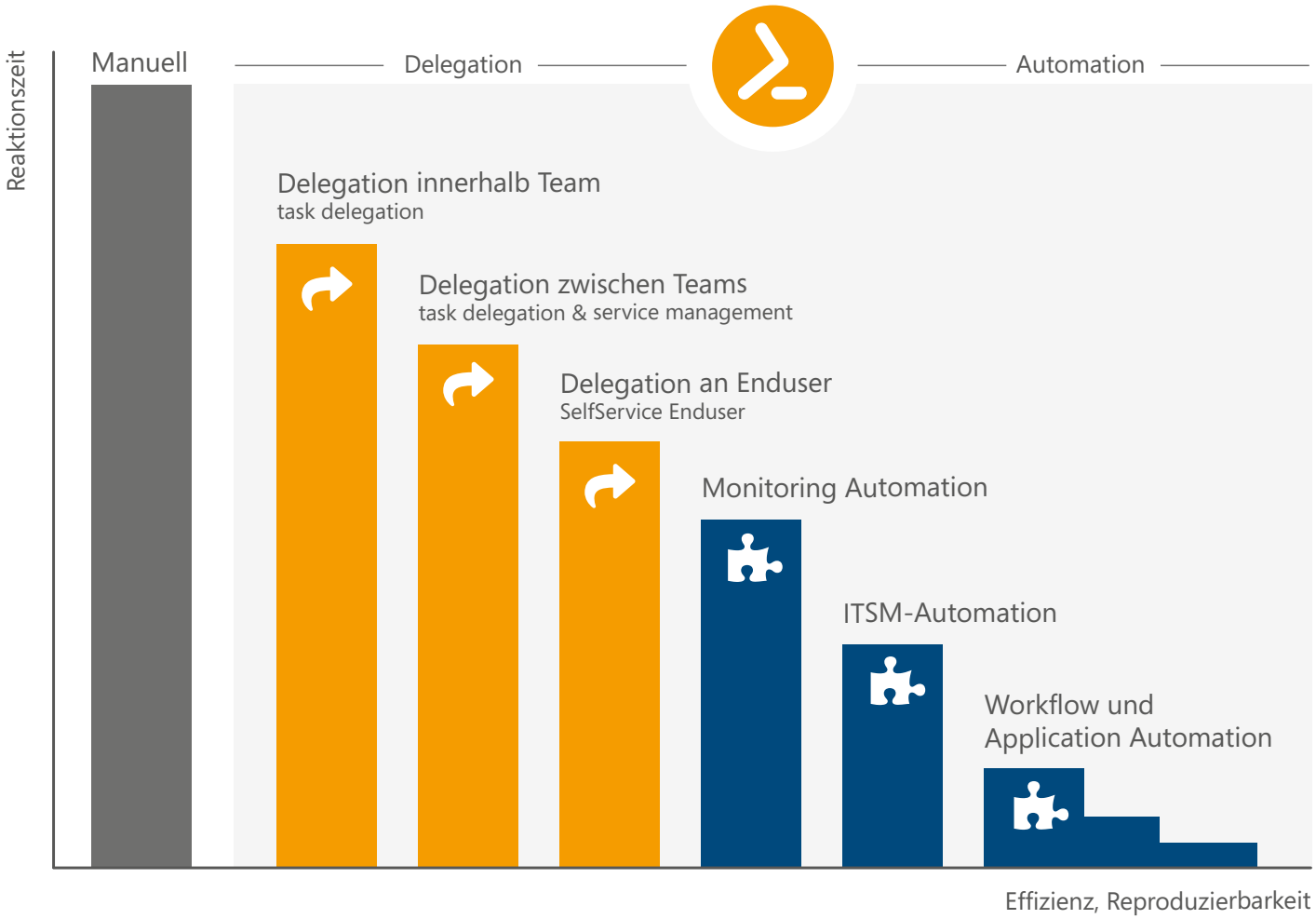
Die Evolution der PowerShell im täglichen Einsatz ist in den letzten Jahren sichtbarer geworden. Nachdem immer mehr Admins der täglichen Klickorgien leid waren, konnten sie mit PowerShell Cmdlets viele Aufgaben auf der Kommandozeile ausführen – immer mehr davon gibt es sogar nur noch dort. Um Aufgabenstellungen in ihrer Gesamtheit besser abbilden zu können, bot sich in einem nächsten Schritt die Verwendung von PowerShell-Scripten an.

Auch diese werden auf der Kommandozeile gestartet, heute oftmals noch lokal, nachdem sich der Administrator per RDP auf das System verbunden hat. Ein weiterer evolutionärer Schritt ist die Verwendung von PowerShell-Remoting. Somit lassen sich Befehle und Scripte auf entfernten Zielsystemen ausführen, ohne sich erst dort am System einloggen zu müssen.

Die PowerShell eignet sich vor diesem Hintergrund perfekt für den Bottom-Up Ansatz, weil Admins damit:

- für ihre tägliche Arbeit schnelle Verbesserungen und Ergebnisse erzielen
- schrittweise persönliche Erfahrungen sammeln können
- es ein kostengünstiger und wenig komplexer Einstieg ist
- nicht jeder Admin sofort zum PowerShell-Profi werden muss

ScriptRunner®



Jeder der genannten Punkte ist eine Voraussetzung und notwendige Erfahrung für eine spätere Automation mit PowerShell.

Essenziell für die eigentliche Automation im IT-Betriebsteam sind dann die weiteren Fragestellungen:

- Wie verwenden Administratoren die Scripte im Team?
- Wie lässt sich eine zentrale Bereitstellung der Scripte sicherstellen?
- Wer darf wann und wie welche Scripte auf welchen Systemen ausführen?
- Wie wird sichergestellt, dass die Scripte nur kontrolliert, richtliniengesteuert und jederzeit sicher auf den Zielsystemen verarbeitet werden?
- Wie transparent ist das Ganze? Wo und wie erfolgt ein zentrales Reporting und welche Audit-Möglichkeiten sind notwendig?
- Wie soll der Entwicklungs- und Testprozess für Scripte aussehen und wie lässt er sich im Betrieb etablieren?

In diesem Kontext könnte zum Beispiel die sichere Delegation von Aufgaben eine der ersten Stufen für eine Automation mit der PowerShell sein.

BEISPIEL USER PROVISIONING

User Provisioning bedeutet vereinfacht, Benutzerkonten in der IT-Infrastruktur und in Unternehmensanwendungen automatisch zu erstellen, zu ändern, zu deaktivieren und zu entfernen. Gleichzeitig sollen dabei auch die passenden Berechtigungen vergeben oder entzogen werden. Die Verwaltung von Benutzerkonten und Berechtigungen ist erfahrungsgemäß zeitraubend. Aber nicht nur das. Auch die Delegation der damit verbundenen Aufgaben etwa an den Service Desk stellt eine Herausforderung dar.

Um eine sichere Delegation zu ermöglichen, können zum Beispiel das Active Directory (AD) und Exchange mit Role Based Access Control (RBAC) versehen werden. Anschließend erfolgt dann eine Schulung der Service-Desk-Anwender für zwei administrative Werkzeuge, die AD User und Computers Console sowie die Exchange Management Console. Das ist jedoch relativ aufwändig und auch unter Sicherheitsgesichtspunkten nicht unproblematisch. So etwa sieht ein wirksames Sicherheits-Schalenmodell einen direkten Zugriff von Service-Desk-Mitarbeitern auf zentrale Systeme gar nicht vor. Die Zugriffe erfolgen stattdessen immer indirekt, kontrolliert und überwacht.

ScriptRunner®

Sehen wir uns dazu ein konkretes Beispiel an:
Es soll ein neuer Benutzer im AD sowie eine Mailbox angelegt, der Zugriff auf Dateien und Drucker berechtigt, die Verteilerlisten aktualisiert und eine erste Mail mit weiteren Informationen versendet werden. Im klassischen Click, Copy & Paste erwarten den Administrator eine Fülle von Fenstern, Tabs und Clicks. Reproduzierbarkeit und Nachvollziehbarkeit sind nicht wirklich gegeben, die Fehleranfälligkeit ist hoch, der Zeitaufwand entsprechend auch.

Nachdem ein Administrator erste Erfahrungen mit PowerShell-Befehlen gemacht hat, könnte ein Script entstehen, das die Einzelaufgaben bündelt. Der Einsatz des Scripts auf der PowerShell Console führt nun zu einer höheren Reproduzierbarkeit mit deutlich weniger Fehlern, ein Logging verbessert auch die Nachvollziehbarkeit. Der Zeitaufwand sinkt merklich. In einem nächsten Schritt könnte der Administrator nach den guten Erfahrungen die Scriptanwendung im Team teilen. Er stellt das Script allen zur Verfügung. Außerdem erklärt sich ein Mitarbeiter aus dem Service Desk bereit, am Experiment mit dem User-Provisioning-Script teilzunehmen.

Nach kurzer Zeit stellen sich jedoch gravierende Probleme ein:

- Es gibt unterschiedliche Versionen des Scripts, weil einige Kollegen persönliche Anpassungen vorgenommen haben.
- Die Ausführung und Ergebnisse der unterschiedlichen Scripte ist nicht mehr reproduzier- und vergleichbar.
- Die Ausführung der Scripte in der persönlichen PowerShell Console verhindert eine einheitliche Protokollierung und damit Nachvollziehbarkeit.
- Einige Kollegen im Team möchten das Script zwar anwenden, fragen jedoch nach einer grafischen Eingabemaske, da sie die Befehlszeile und die vielen notwendigen Parameter für den Aufruf nicht immer sofort parat haben.
- Bei der Anwendung des Scripts durch den Mitarbeiter im Service Desk stellt sich heraus, dass er zu viele administrative Berechtigungen bräuchte. Alternativ könnten diese Berechtigungen nur im Script-Code hinterlegt werden. Ein viel zu hohes Sicherheitsrisiko.
- Zu alledem wurde das Script inzwischen weiterentwickelt und verbessert. Nötig wäre also ein Prozess, der Entwicklung, Test und Überführen von Scripten in die Produktion sicherstellt.

Bei genauer Betrachtung zeigen sich die zwei Seiten der Medaille: Auf der Pro-Seite ist zu vermerken, dass ein Anfang für die Automation gemacht ist und das PowerShell-Script für das komplexe User Provisioning reproduzierbare Ergebnisse liefert. Die Kehrseite zeigt, dass die Anwendung der Lösung nur als Ein-Mann-Prozess reibungslos zu funktionieren scheint. Die Verwendung im Team scheitert an organisatorischen und technischen Hürden, an eine sichere Delegation ist nicht zu denken.

Der Administrator könnte in einem nächsten Schritt versuchen, die Scripte auf einem gemeinsamen Verzeichnis zu speichern. Die Erfahrungen sind aber oft eher schlecht, die Anzahl der persönlichen Varianten nimmt meist nicht wirklich ab. Auch um professioneller entwickeln zu können, fällt die Entscheidung häufig zugunsten Visual Studio Code mit einem zentralen Git. Versucht ein Administrator mit Bordmitteln auch grafische Benutzerelemente für seine Kollegen zu implementieren, stellt er meist fest, dass das viel zu kompliziert ist und sich die Erfordernisse für den Service Desk darüber hinaus so nicht erfüllen lassen.

Stattdessen müsste er jetzt kompilierte EXE-Dateien an die einzelnen Mitarbeiter verteilen. Deren Ausführung verlangt allerdings die gleichen administrativen Rechte und Randbedingungen wie ein Script. Außerdem lassen sich die Scripte immer noch nicht zentralisiert ausführen. Die Ausführungsberechtigungen stellen weiterhin ein Problem dar, ein zusammenhängendes und übergreifendes Reporting ist mit den verteilten Logdateien nicht handhabbar. Eine Richtlinienvorgabe, welches Script wie und wo ausgeführt werden soll, ist unmöglich.

ScriptRunner®

KOMPONENTEN EINER AUTOMATIONSLÖSUNG

Die beschriebenen Erfahrungen sind oft der Normalfall beim Eintritt in die Automationswelt mit der PowerShell. Es empfiehlt sich daher der gezielte Einsatz einer Automationslösung für die PowerShell.

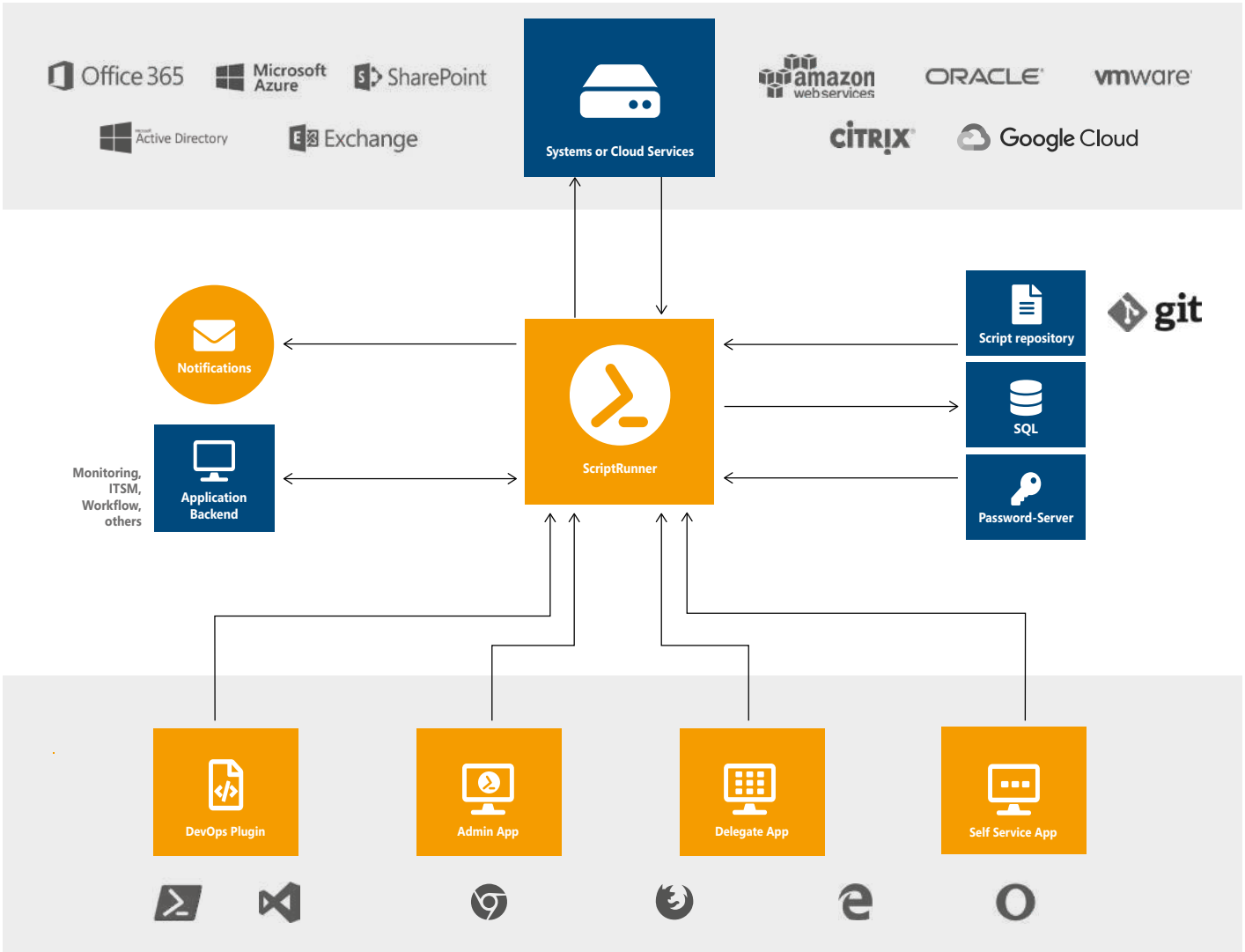
Folgende grundlegende Anforderungen beziehungsweise Bestandteile einer solchen Lösung lassen sich für diese ableiten:

1. Scripte und zum späteren Zeitpunkt auch selbstgeschriebene Module: Diese implementieren die erforderlichen Schritte zur Erledigung einfacher und komplexer Aufgaben. Immer wieder notwendige Funktionen lassen sich in einer Bibliothek von Funktionsscripten oder eigenen Modulen bündeln, um eine Wiederverwendbarkeit von PowerShell Code ohne Copy & Paste zu ermöglichen.

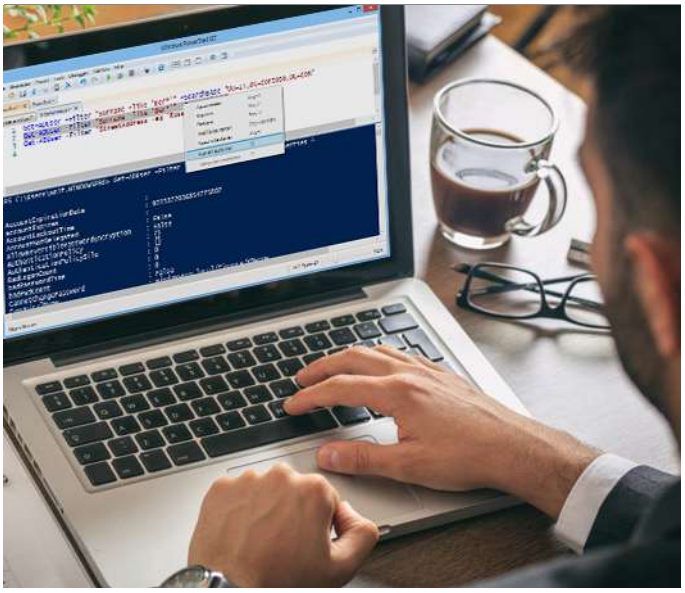
2. Zentrales Script-Repository: Eine zentrale Ablage von Scripten sollte sicherstellen, dass immer wieder das gleiche Script für die gleiche Aufgabe Verwendung findet. Per Definition ist es untersagt, personengebundene Inkarnationen eines Scripts zu erstellen und zu verwenden.

3. Richtlinien für die Ausführung: Die Richtlinien für die Ausführung von Scripten sollten festlegen, welches Script wie und auf welchem Zielsystem in welchem Rechtekontext ausgeführt werden darf, um die Vergleichbarkeit von Ergebnissen zu ermöglichen.

4. Rechtesteuerung für die Ausführung: Die Rechtesteuerung definiert verschiedene Sachverhalte. Welche Credentials werden zur Ausführung des Scripts benötigt? Welche zusätzlichen Credentials müssen zur Verarbeitung in den PowerShell-Prozess eingesteuert werden und wer ist überhaupt berechtigt, eine Scriptaktion zu starten?



ScriptRunner®



5. Ausführungsinstanz für Scripte: Diese Instanz stellt die Reproduzierbarkeit sicher, das heißt die Scripte werden mittels der Script Policies und der Rechtesteuerung immer auf die gleiche Art und Weise auf den Zielsystemen und Services ausgeführt.

6. Zentrales Reporting: Werden Scripte ausgeführt, so ist es wichtig zu wissen, welche Ergebnisse und Fehler dabei aufgetreten sind. Mit einer einheitlichen Reporting-Datenbank, die jede Ausführung zentral protokolliert, lässt sich jederzeit nachvollziehen, was genau passiert ist. Eine wichtige Voraussetzung für Audits.

7. Delegationsmöglichkeit: Vor dem Hintergrund einer zunehmenden Arbeitsteiligkeit und den Eigenheiten einer Administration per Kommandozeile wird klar, dass viele Mitarbeiter im IT-Betrieb so schnell keine PowerShell-Profis werden oder auch sein wollen. Ein Delegationsmechanismus sollte es ermöglichen, einzelne Scriptaktionen auf verschiedene Benutzergruppen rollenbasiert zuzuweisen. Über diese Zuweisung haben PowerShell-Administratoren die Möglichkeit, Scriptaktionen auch durch Dritte sicher ausführen zu lassen.

8. Grafische Anwendung: Obwohl die PowerShell eine Scriptsprache ist, bietet sich für die Einrichtung von Richtlinien, Rechten und Delegation bis hin zum Einsehen der Reports eine grafische Bedienoberfläche als Browser-App an. Um die Anwendung von Scripten in der Breite verfügbar zu machen, ist eine grafische Browser-App für Service-Desk-Mitarbeiter und Administratoren ideal. Über diese können die PowerShell-Parameter eingegeben und validiert sowie eine Scriptaktion gestartet werden.

9. Schnittstellen: Eine Automationslösung für PowerShell steht in der Regel nicht allein im Feld. Monitoring, ITSM, Workflowsysteme und andere sind ebenfalls wichtige Bestandteile der IT-Infrastruktur. Für eine zukünftige Gesamt-Automationslösung ist es enorm wichtig, entsprechende Schnittstellen zur Integration zu bieten. Als Schnittstellenstandard bieten sich Web Services, insbesondere REST an.

10. Entwicklungsumgebung für Scripte: Administratoren, die PowerShell-Scripte entwickeln, sehen sich mit neuen Herausforderungen konfrontiert. Diese teilen sich in zwei Komplexe. Zum einen ist da das Entwicklungswerkzeug inklusive Versionsverwaltung sowie die Gestaltung des Prozesses vom Entwurf bis zur Produktivstellung. Während es am Markt eine Auswahl an technischen Werkzeugen gibt, wie beispielsweise Visual Studio Code und Git, ist die Prozessgestaltung stark von internen organisatorischen Randbedingungen abhängig. Grundsätzlich sind die unterschiedlichen Phasen Entwickeln, Testen und Produktion abzubilden.

FAZIT

Es ist sinnvoll, frühzeitig in das PowerShell-Universum einzusteigen, um die Konzepte, Zusammenhänge und Details besser verstehen zu lernen. Mit der PowerShell bieten sich sehr viele Möglichkeiten für die Automation im IT-Betrieb. Die Flexibilität der Scriptsprache in Verbindung mit der Funktionalität von Befehlen und Modulen erlaubt, die täglichen Aufgaben deckungsgenau in Scripten abzubilden. Für eine sichere Gesamtlösung für alle Mitarbeiter im IT-Team und darüber hinaus sind zusätzliche technische und organisatorische Unterstützungsfunktionen hilfreich, sinnvollerweise mit einer umfänglichen Plattformlösung.

Frank Kresse ist General Manager von der ScriptRunner Software GmbH und Entwickler von ScriptRunner.

Dieser Artikel erschien auch online auf it-administrator.de im Mai 2018

ScriptRunner Software GmbH
Ludwig-Erhard-Straße 2
76275 Ettlingen
Germany

Tel: +49 (0) 7243 20715-0
Fax: +49 (0) 7243 20715-99
E-Mail: info@scriptrunner.com
Web: www.scriptrunner.com