Automatisierung per REST-API und PowerShell



WISSEN

Alles zum Kemp LoadMaster mit Microsoft-Brille: Exchange, SharePoint, ADFS, RDS, Eigenentwicklungen und Kerberos an einem Ort.

› **Kemp LoadMaster**

BERATUNG

Du willst Wartungsfenster, Zertifikatswechsel und Dokumentation skripten, ohne dass das Skript zum neuen Risiko wird? Wir bauen es mit dir, inklusive Rechtekonzept.

› **Beratung zum Kemp LoadMaster**

SCHULUNG

API, PowerShell-Modul und typische Betriebsaufgaben im Labor üben, wo ein falscher Parameter nur ein Lacher ist und kein Ticket.

› **Kemp-LoadMaster-Schulung**

Jeder Patchday beginnt am LoadMaster mit demselben Ritual: Anmelden an der Weboberfläche, Virtual Service suchen, Real Server deaktivieren, Exchange patchen, zurück zur Weboberfläche, Real Server aktivieren, nächster Server. Bei vier Exchange-Servern, zwei ADFS-Knoten und einer RDS-Farm sind das schnell vierzig Klicks pro Server, und irgendwann gegen Mitternacht vergisst jemand das Aktivieren. Der Server bleibt dann so lange außen vor, bis die anderen unter Last einknicken. Das ist der Moment, in dem man feststellt, dass N+1 nur auf dem Papier existierte.

Die gute Nachricht: Der LoadMaster lässt sich vollständig per REST-API steuern, und Progress liefert dafür ein PowerShell-Modul. Wer als Microsoft-Administrator ohnehin Exchange per Exchange Management Shell und Server per PowerShell pflegt, fühlt sich dort schnell zu Hause. Die schlechte Nachricht: Ein Skript ist ein sehr fleißiger, sehr dummer Kollege. Es führt Fehler mit derselben Gründlichkeit aus wie richtige Anweisungen, nur schneller und auf allen Geräten gleichzeitig.

Dieser Beitrag aus der Serie zum Kemp LoadMaster zeigt dir, wie du die API freischaltest und absicherst, wie du mit dem PowerShell-Modul arbeitest und wie drei typische Aufgaben aussehen: Real Server in Wartung nehmen, Zertifikat austauschen und Konfiguration dokumentieren. Und er zeigt dir, wo Automatisierung zur Falle wird, nämlich dann, wenn sie manuelle Änderungen still überschreibt. „Kemp LoadMaster PowerShell“ klingt nach einem Suchbegriff, ist aber eigentlich ein Betriebskonzept.

FAKTEN · Das Wichtigste in 30 Sekunden

Die API-Schnittstelle ist ab Werk ausgeschaltet. Du aktivierst sie in der Weboberfläche im Bereich Certificates & Security unter Remote Access mit der Option Enable API Interface.

Es gibt zwei Formen: API v1 über /access/<Befehl> mit Parametern in der URL und XML-Antwort, sowie API v2 als POST mit JSON-Körper an /accessv2.

Anmelden kannst du dich per Benutzer und Kennwort, per Client-Zertifikat (nur v1, laut Hersteller für eine spätere Abkündigung vorgesehen) oder per API-Schlüssel. Progress empfiehlt API-Schlüssel.

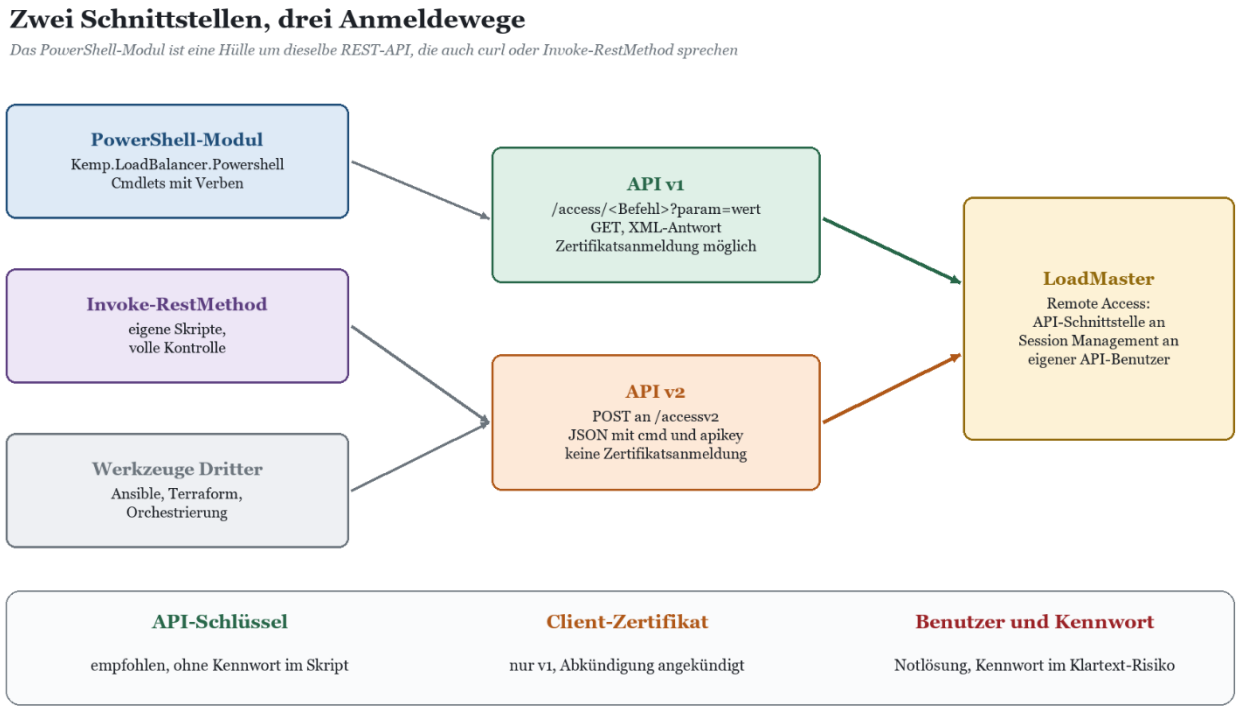
Der Standardbenutzer bal hat über die API vollen Zugriff und kann keine Zertifikatsanmeldung nutzen. Für Skripte gehört ein eigener Benutzer mit genau den nötigen Rechten her.

Mehr als 30 Aufrufe in weniger als drei Sekunden, gezählt über alle API-Schnittstellen, beendet der LoadMaster mit einem Verbindungsabbruch. Schleifen brauchen also eine Bremse.

Das PowerShell-Modul heißt Kemp.LoadBalancer.Powershell und ist seit Firmware 7.2.36 signiert.

API freischalten und absichern: die Tür nur für den Richtigen öffnen

Die API ist kein Zusatzprodukt, sondern eine zweite Tür in dieselbe Verwaltungsebene, durch die du sonst die Weboberfläche betrittst. Wer durch diese Tür kommt, kann Virtual Services löschen, Zertifikate austauschen und Firmware einspielen, je nach Rechten. Deshalb gilt hier dieselbe Regel wie für die Verwaltung von Domänencontrollern: Die Tür ist nur aus dem Verwaltungsnetz erreichbar, und wer durchgeht, hat einen Namen.



Skizze: Aufbau nach Herstellerdokumentation; Befehlsumfang abhängig von Firmware und Lizenz

Schnittstelle aktivieren und Session Management einschalten

Die API aktivierst du in der Weboberfläche unter Certificates & Security, Remote Access, mit der Option Enable API Interface. Für die JSON-Variante der API und für die Anmeldung per Client-Zertifikat verlangt der Hersteller zusätzlich, dass Session Management unter Certificates & Security, Admin WUI Access eingeschaltet ist. Für JSON muss außerdem die Option Require Basic Authentication auf derselben Seite deaktiviert sein. Session Management ist ohnehin eine gute Idee: Es bringt Sitzungs-Timeouts, Sperren nach Fehlversuchen und sauberere Protokolle mit sich.

Wichtiger als das Häkchen ist die Frage, von wo aus die API erreichbar ist. Die API lauscht auf derselben Verwaltungsadresse wie die Weboberfläche. Wenn du die Verwaltung bereits auf ein eigenes Management-Netz gelegt und per Firewall auf die Admin-Arbeitsplätze und den Automatisierungsserver beschränkt hast, ist die API automatisch mit geschützt. Wenn die Weboberfläche dagegen aus dem gesamten Client-Netz erreichbar ist, hast du jetzt zwei Probleme statt einem.

WARNUNG · Die API niemals über einen öffentlichen Virtual Service erreichbar machen

Es kommt vor, dass jemand die Verwaltungsoberfläche über einen Virtual Service ins Internet veröffentlicht, um von unterwegs nachsehen zu können. Damit liegt auch die API offen, und ein erratenes Kennwort reicht für die vollständige Kontrolle über Exchange-, ADFS- und RDS-Veröffentlichungen. Fernzugriff gehört über VPN oder einen Sprungserver geführt, nicht über das Gerät, das du gerade verwaltest.

Ein eigener Benutzer pro Zweck, nicht bal für alles

Der Standardbenutzer bal ist so etwas wie der Domänen-Administrator des LoadMasters: Er darf alles, er ist überall gleich, und man sieht im Protokoll nicht, wer ihn gerade benutzt. Über die API hat bal vollen Zugriff, alle anderen Benutzer nur die Rechte, die du ihnen in der lokalen Benutzerverwaltung zuweist. Diese Rechte lassen sich übrigens nicht über die API selbst ändern. Ein kompromittiertes Skript kann sich also nicht selbst befördern.

In der Praxis bewährt sich eine Aufteilung nach Zweck. Ein Konto liest nur und erzeugt die nächtliche Dokumentation. Ein zweites darf Real Server aktivieren und deaktivieren und dient den Wartungsskripten. Ein drittes darf Zertifikate verwalten. Wer schon einmal ein Rollenkonzept für Exchange RBAC gebaut hat, kennt das Prinzip. Wie fein die Rechte im Einzelnen geschnitten werden können, hängt vom Firmware-Stand ab. Prüfe in der Benutzerverwaltung, welche Berechtigungsstufen dein Gerät anbietet, und vergib im Zweifel weniger.

Konto (Beispiel)	Zweck	Rechte	Anmeldung	Wer kennt den Schlüssel
api-doku	Nächtliche Dokumentation, Zertifikatsbericht	nur lesen	API-Schlüssel	Automatisierungsserver, Tresor
api-wartung	Real Server für Patchday aktivieren und deaktivieren	Real Server ändern	API-Schlüssel	Automatisierungsserver, Tresor
api-zert	Zertifikate einspielen und ersetzen	Zertifikate verwalten	API-Schlüssel oder Client-Zertifikat	PKI-Team, Tresor
bal	Erstinstallation, Notfall	alles	Kennwort, nicht für Skripte	Notfallumschlag, Tresor

TIPP · Benutzernamen, die man im Protokoll versteht

Nenne die API-Konten nach ihrer Aufgabe, nicht nach der Person, die das Skript geschrieben hat. Wenn das Syslog um drei Uhr nachts eine Anmeldung von api-wartung zeigt, weißt du sofort, welches Skript du dir ansehen musst. Bei einer Anmeldung von tmueller weißt du nur, dass jemand im Urlaub ist.

API-Schlüssel, Zertifikat oder Kennwort: welcher Anmeldeweg wofür

Mit Benutzer und Kennwort funktioniert alles, aber das Kennwort muss irgendwo stehen, und meist steht es dann im Skript, in einer Aufgabenplanung oder in einer Datei mit dem sprechenden Namen pw.txt. Die Anmeldung per Client-Zertifikat war lange der saubere Weg: Der LoadMaster erzeugt für einen lokalen Benutzer ein Zertifikat, du wandelst es in eine PFX-Datei um, importierst es auf dem Automatisierungsserver in den Zertifikatsspeicher und das Skript meldet sich ohne Kennwort an. Der Haken: Die Zertifikatsanmeldung funktioniert nur mit der API v1, und Progress kündigt in der Dokumentation an, sie zu einem späteren Zeitpunkt abzukündigen.

Damit ist der API-Schlüssel der Weg für neue Skripte. Er wird einem Benutzer zugeordnet, erbt dessen Rechte und ersetzt Benutzername und Kennwort im Aufruf. Wird in der API v2 ein Schlüssel mitgegeben, hat er Vorrang vor Benutzer und Kennwort. Der Schlüssel ist natürlich ebenfalls ein Geheimnis, aber eines, das du gezielt widerrufen kannst, ohne das Kennwort des Kontos zu ändern. Ablegen solltest du ihn dort, wo du auch andere Geheimnisse für Skripte ablegst, etwa im Modul Microsoft.PowerShell.SecretManagement mit einem Tresor deiner Wahl.

Anmeldeweg	API v1	API v2 (JSON)	Stärke	Schwäche
Benutzer und Kennwort	ja	ja	funktioniert immer	Kennwort im Skript, Rotation betrifft alle Skripte
Client-Zertifikat	ja	nein	kein Kennwort, Schlüssel im Zertifikatsspeicher	Abkündigung angekündigt, nicht für bal
API-Schlüssel	ja	ja	vom Hersteller empfohlen, einzeln widerrufbar	Schlüssel braucht einen sicheren Ablageort

WICHTIG · Wenn du heute Zertifikatsanmeldung nutzt

Kein Grund zur Panik, aber ein Grund für einen Eintrag in der Planung. Prüfe vor jedem Firmware-Update in den Release Notes, ob sich an der Zertifikatsanmeldung etwas ändert, und stelle neue Skripte gleich auf API-Schlüssel um. Wie du Updates ohne Überraschungen planst, steht in

linkhttps://www.boddenberg.de/kemp-firmware-update/Firmware-Updates, Backup und Notfallplan: LoadMaster-Betrieb ohne Herzrasen

.

Das PowerShell-Modul von Kemp: Cmdlets statt URL-Bastelei

Die REST-API selbst ist einfach, aber nicht schön. Ein Befehl der API v1 sieht aus wie eine Adresse mit angehängten Parametern, die Antwort kommt als XML. Das lässt sich mit Invoke-RestMethod gut verarbeiten, aber du schreibst dabei viel Klebstoff: Adressen zusammensetzen, Sonderzeichen codieren, Fehlercodes aus dem XML fischen. Das PowerShell-Modul nimmt dir das ab und liefert Cmdlets mit vertrauten Verben wie Get, New, Set, Enable und Disable.

Installieren, signieren, verbinden

Das Modul liegt in der PowerShell Gallery unter dem Namen Kemp.LoadBalancer.Powershell. Seit Firmware 7.2.36 ist es signiert. Je nach Ausführungsrichtlinie musst du das Signaturzertifikat von Progress als vertrauenswürdigen Herausgeber importieren, bevor das Modul lädt. Installiere es auf dem Automatisierungsserver, nicht auf jedem Admin-Notebook: Dann gibt es genau eine Modulversion, genau einen Ort mit Geheimnissen und genau ein Protokoll.

PowerShell

```
# Modul installieren (einmalig, auf dem Automatisierungsserver)
```

```
Install-Module -Name Kemp.LoadBalancer.Powershell -Scope AllUsers
Import-Module Kemp.LoadBalancer.Powershell

# Welche Cmdlets gibt es in dieser Version?
Get-Command -Module Kemp.LoadBalancer.Powershell | Sort-Object Noun | Format-Table Name
Get-Help Initialize-LmConnectionParameters -Full
```

Damit du nicht bei jedem Cmdlet Adresse und Anmeldedaten mitgeben musst, legst du mit Initialize-LmConnectionParameters einmal pro Sitzung fest, mit welchem LoadMaster du sprichst. Das Cmdlet nimmt wahlweise Anmeldedaten oder die Angaben zum Client-Zertifikat entgegen. Test-LmServerConnection prüft danach, ob das Gerät erreichbar ist.

PowerShell

```
# Variante A: Anmeldedaten aus dem Tresor (SecretManagement)
$cred = Get-Secret -Name 'LM-api-wartung'          # liefert ein PSCredential
Initialize-LmConnectionParameters -Address '<LM-Verwaltungsadresse>' -LBPort 443 -Credential
$cred

# Variante B: Client-Zertifikat aus dem Zertifikatsspeicher (nur API v1)
Initialize-LmConnectionParameters -Address '<LM-Verwaltungsadresse>' -LBPort 443 `
    -SubjectCN 'api-wartung' -CertificateStoreLocation 'Cert:\CurrentUser\My'

Test-LmServerConnection -ComputerName '<LM-Verwaltungsadresse>' -Port 443
```

TIPP · Parameternamen sind versionsabhängig

Das Modul wächst mit der Firmware, und Parameter heißen nicht in jeder Version gleich. Bevor du ein Beispiel aus diesem Artikel oder aus einem Forum übernimmst, prüfst du es mit Get-Help und den Parametern -Full oder -Examples gegen deine installierte Version. Das gilt besonders für die Unterstützung von API-Schlüsseln im Modul. Ältere Modulversionen kennen sie nicht.

Wenn das Modul nicht reicht: die API direkt ansprechen

Nicht jeder API-Befehl hat ein Cmdlet, und manchmal willst du genau sehen, was über die Leitung geht. Dann sprichst du die API v2 direkt mit Invoke-RestMethod an. Der Aufruf ist immer ein POST an /accessv2 mit einem JSON-Objekt, dessen einziges Pflichtfeld cmd ist. Parameter, die ein Befehl nicht braucht, ignoriert der LoadMaster. Einige wenige Befehle, etwa das Einspielen von Patches, funktionieren laut Hersteller nur über die API v1.

PowerShell

```
# API v2 direkt: Firmware-Stand und Statistik lesen
$apiKey = Get-Secret -Name 'LM-api-doku' -AsPlainText
$uri    = 'https://<LM-Verwaltungsadresse>/accessv2'

$body = @{ apikey = $apiKey; cmd = 'stats' } | ConvertTo-Json
Invoke-RestMethod -Method Post -Uri $uri -Body $body -ContentType 'application/json'
```

WARNUNG · Zertifikatsprüfung nicht abschalten

Viele Beispiele im Netz arbeiten mit -SkipCertificateCheck oder dem curl-Schalter -k, weil die Verwaltungsoberfläche ein selbst signiertes Zertifikat zeigt. Für ein Skript, das einen API-Schlüssel mit Schreibrechten überträgt, ist das eine Einladung zum Mitlesen. Stelle der Verwaltungsoberfläche ein Zertifikat aus deiner internen PKI aus, etwa für lm-mgmt.contoso.local, und lass die Prüfung an. Mehr zum Zusammenspiel von PKI und LoadMaster findest du in

linkhttps://www.boddenberg.de/kemp-tls-offload-zertifikate/TLS-Offload, Re-Encryption und

Zertifikate: PKI trifft LoadMaster

.

FAKTEN · Grenzen der API, die du kennen solltest

Ein Aufruf enthält genau einen Befehl. Mehrere Änderungen bedeuten mehrere Aufrufe, und zwischen ihnen gibt es keine Transaktion. Bricht das Skript in der Mitte ab, bleibt ein halber Zustand stehen.

Eine URL der API v1 darf höchstens etwa 1 KB lang sein und keine Leerzeichen enthalten. Längere Inhalte wie Zertifikate gehen per POST.

Mehr als 30 Aufrufe in unter drei Sekunden führen zum Verbindungsabbruch. Eine kurze Pause in Schleifen über viele Virtual Services ist keine Kosmetik.

Nicht jeder Befehl steht auf jedem Gerät zur Verfügung. Manche hängen von der Lizenz ab, etwa WAF- oder ESP-Funktionen.

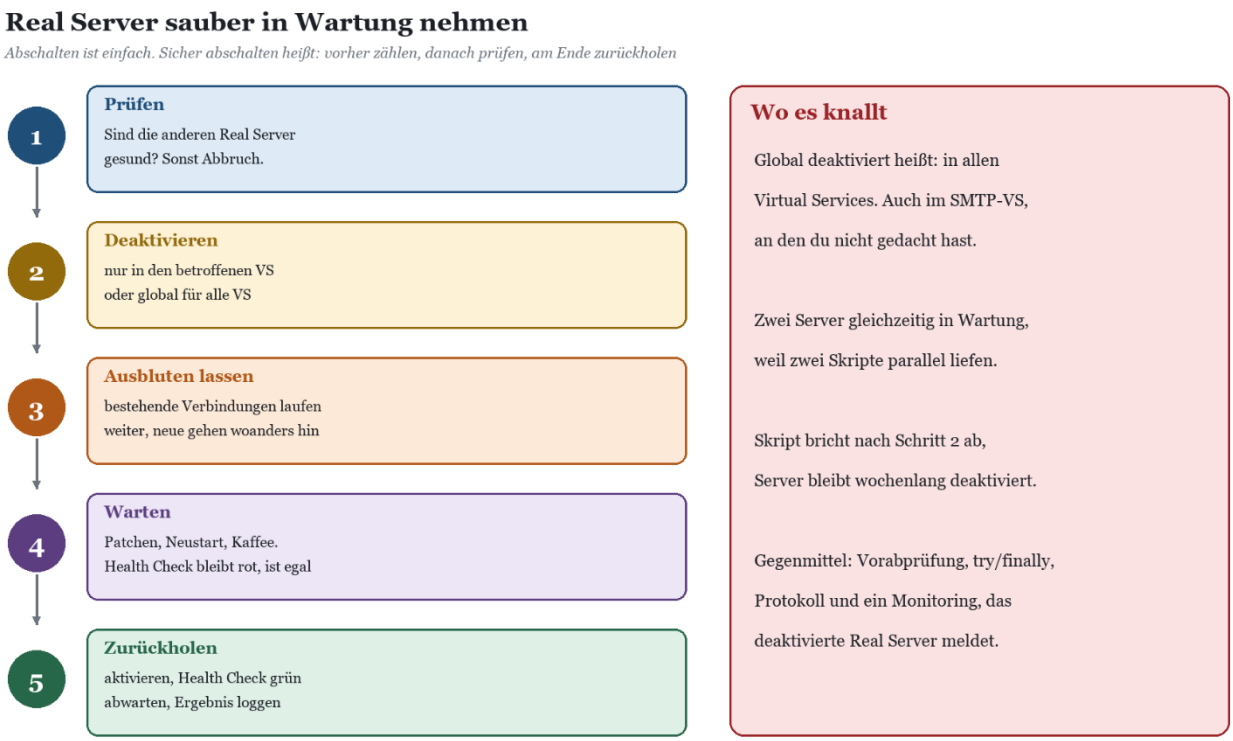
Fehler meldet die API mit einem Statuscode, der dem HTTP-Status entspricht, plus Klartextmeldung. Werte beides aus, statt nur auf eine Ausnahme zu warten.

Drei typische Aufgaben: Wartung, Zertifikat, Dokumentation

Die folgenden Beispiele sind bewusst kurz. Sie zeigen das Prinzip, nicht das fertige Produkt. Alles in spitzen Klammern ist ein Platzhalter. Bevor du ein Skript produktiv einsetzt, läuft es im Labor oder wenigstens gegen einen Virtual Service, an dem nichts Wichtiges hängt. Wer ein HA-Paar betreibt, spricht immer die gemeinsame Verwaltungsadresse an, damit die Änderung auf der aktiven Einheit landet und synchronisiert wird. Details dazu stehen in HA-Paar, Failover und Konfig-Sync: zwei LoadMaster, ein Gedanke.

Real Server in Wartung nehmen

Die häufigste Aufgabe und gleichzeitig die mit dem größten Überraschungspotenzial. Es gibt zwei Arten, einen Real Server zu deaktivieren: in einem bestimmten Virtual Service oder global, also in allen Virtual Services, in denen dieselbe Adresse als Real Server eingetragen ist. Global ist bequem für den Patchday eines Exchange-Servers, der in HTTPS-, SMTP- und IMAP-Diensten steckt. Global ist aber auch gefährlich, wenn derselbe Server unerwartet in einem weiteren Virtual Service hängt, etwa einem alten Test-VS, den niemand mehr kennt. Die Begriffe Virtual Service und Real Server erklärt Virtual Service, Real Server, SubVS: das Grundvokabular des LoadMasters im Detail.



Skizze: Ablauf als Empfehlung; Verhalten bestehender Verbindungen abhängig von Persistenz und Protokoll

Ein brauchbares Wartungsskript prüft deshalb zuerst, ob die übrigen Real Server gesund sind. Ist der Partner gerade selbst rot, bricht es ab, statt den letzten funktionierenden Server abzuschalten. Ob der Health Check wirklich etwas Sinnvolles prüft, ist eine eigene Frage, die Health Checks, die wirklich prüfen: warum „Port 443 offen“ nicht reicht beantwortet.

PowerShell

```
# Real Server global deaktivieren, mit Vorabprüfung und garantiertem Zurückholen
$rs = '<IP-Adresse-des-Real-Servers>'

$vs = Get-AdcVirtualService                        # alle Virtual Services lesen
# Vorabprüfung: Hat jeder betroffene VS noch mindestens einen anderen aktiven, gesunden Real
Server?
# (Auswertung der Rückgabe je nach Modulversion anpassen)

try {
    Disable-AdcRealServer -RSIpaddress $rs         # global: wirkt in allen VS mit dieser
Adresse
    Write-Host "Real Server $rs deaktiviert. Jetzt patchen."
    Read-Host 'Weiter mit Enter, wenn der Server wieder läuft'
}
finally {
    Enable-AdcRealServer -RSIpaddress $rs          # läuft auch, wenn oben etwas schiefgeht
    Write-Host "Real Server $rs wieder aktiviert. Health Check abwarten."
}
```

Der finally-Block ist das Wichtigste an diesem Beispiel. Er sorgt dafür, dass der Server auch dann zurückkommt, wenn das Skript zwischendurch abbricht oder jemand Strg+C drückt. In einem unbeaufsichtigten Ablauf, etwa in einer Orchestrierung für Cluster-Aware Updating oder einem Runbook, ersetzt du das Read-Host durch eine echte Prüfung, ob der Server nach dem Neustart wieder Dienste anbietet.

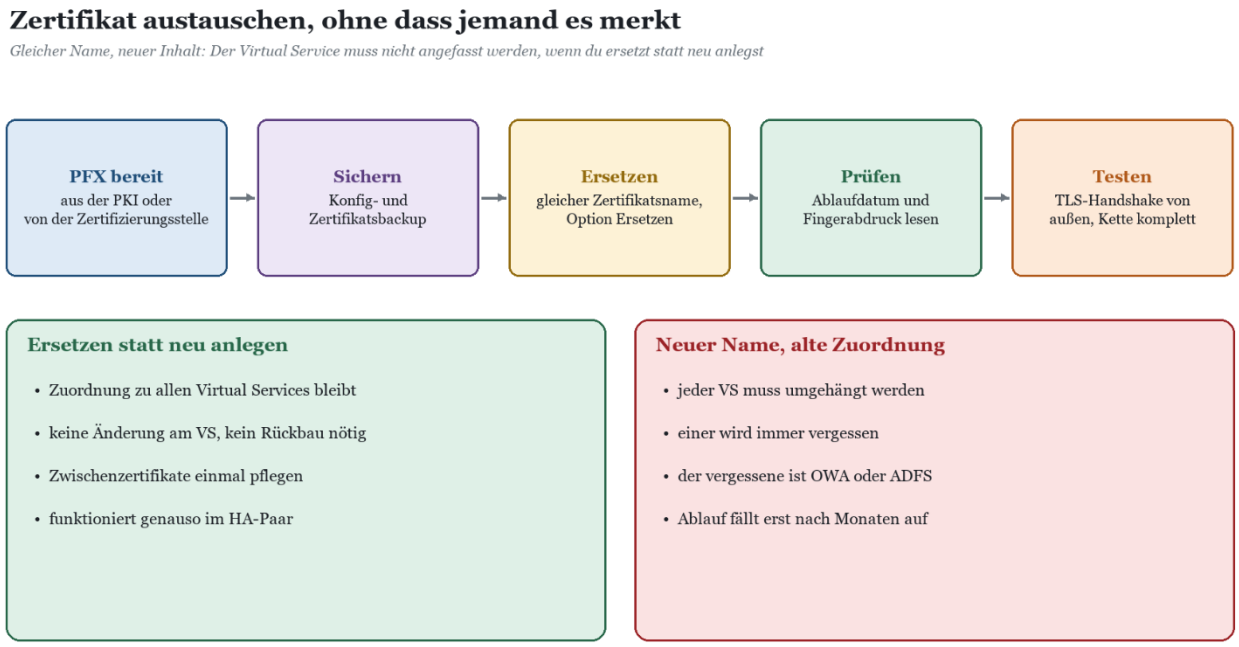
AUS DER PRAXIS · Der Patchday, der zwei Wochen dauerte

Typischer Verlauf: Ein Wartungsskript deaktiviert einen Exchange-Server global, das Patchen läuft, der Server startet neu. Das Skript läuft aber als geplante Aufgabe, die nach dem Neustart des Automatisierungsservers nicht fortgesetzt wird, denn der wurde am selben Abend gleich mitgepatcht. Der Exchange-Server ist danach gesund, aber am LoadMaster deaktiviert. Zwei Wochen später fällt der zweite Server aus, und plötzlich steht Exchange. Seitdem meldet das Monitoring jeden deaktivierten Real Server, der länger als vier Stunden in diesem Zustand ist.

Genau so eine Meldung bekommst du übrigens einfach hin, wenn der LoadMaster bereits im Monitoring steckt. Wie das geht, steht in Monitoring, Syslog und SNMP: den LoadMaster ins Monitoring holen.

Zertifikat austauschen

Zertifikate laufen ab, und seit die Laufzeiten öffentlicher TLS-Zertifikate schrittweise verkürzt werden, laufen sie immer öfter ab. Wer vier oder fünf Mal im Jahr dasselbe Zertifikat in die Weboberfläche lädt, wird das irgendwann vergessen. Der Trick beim Skripten ist derselbe wie beim Klicken: Du ersetzt das bestehende Zertifikat unter gleichem Namen, statt ein neues anzulegen. Dann bleibt die Zuordnung zu allen Virtual Services erhalten, und du musst keinen einzigen VS anfassen.



Skizze: Ablauf für den Wechsel eines Serverzertifikats; private Schlüssel nie im Skript oder Repository ablegen

PowerShell

```
# Bestehendes Zertifikat unter gleichem Namen ersetzen
$pfxPfad = '<Pfad>\mail.contoso.de-2026.pfx'
$pfxPw   = Get-Secret -Name 'PFX-mail-contoso' -AsPlainText

Get-TlsCertificate                                  # Ist-Zustand: Name und Ablaufdatum notieren

New-TlsCertificate -Name '<Zertifikatsname-am-LM>' -Path $pfxPfad -Password $pfxPw -Replace

Get-TlsCertificate                                  # Kontrolle: neues Ablaufdatum sichtbar?
```

Nach dem Austausch prüfst du nicht am LoadMaster, sondern von außen, was ein Client sieht: Ablaufdatum, ausstellende Zertifizierungsstelle und die vollständige Kette mit Zwischenzertifikat.

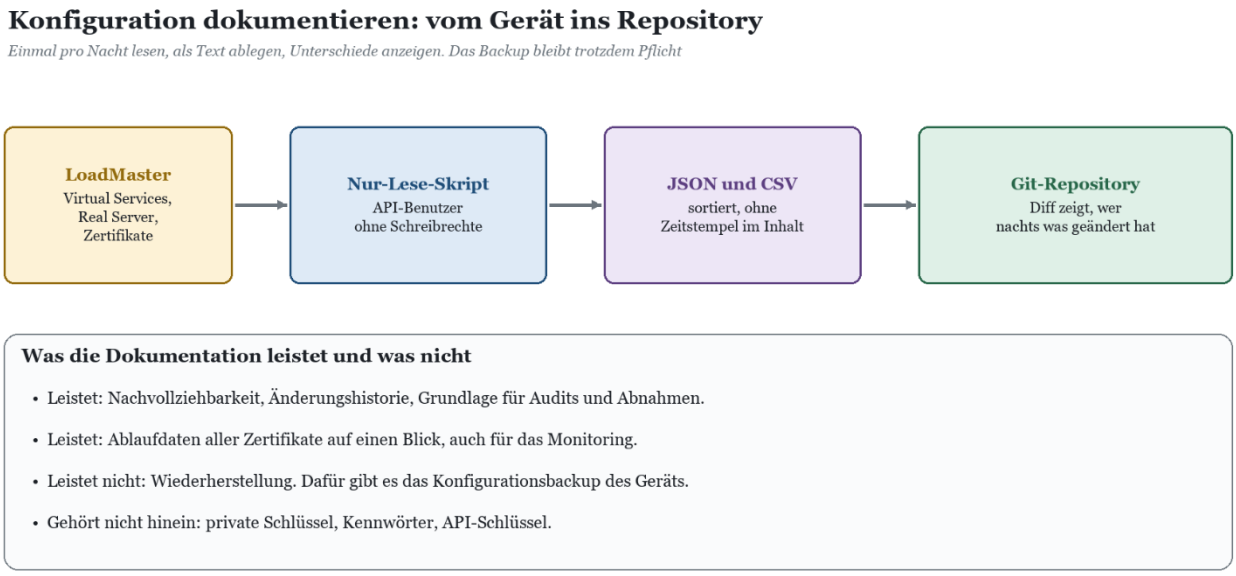
Ein fehlendes Zwischenzertifikat fällt in Edge und Chrome oft nicht auf, weil die Browser es nachladen, wohl aber bei Outlook auf dem Smartphone oder bei einem Partner-Mailserver, der über TLS einliefert.

WARNUNG · Private Schlüssel haben im Skript nichts verloren

Die PFX-Datei enthält den privaten Schlüssel. Sie gehört weder in ein Git-Repository noch dauerhaft auf einen Dateiserver. Lege sie nur für die Dauer des Austauschs auf dem Automatisierungsserver ab, hole die Passphrase aus dem Tresor und lösche die Datei danach. Wer den Ablauf vollständig automatisieren will, etwa aus einer internen Active Directory Certificate Services oder über ACME, braucht ein eigenes Konzept für den Transport des Schlüssels, nicht nur ein Skript.

Konfiguration dokumentieren

Die nützlichste Automatisierung ist die, die nichts ändert. Ein nächtliches Skript, das alle Virtual Services, Real Server, Health-Check-Einstellungen und Zertifikate liest und als Textdateien in ein Git-Repository schreibt, beantwortet Fragen, die sonst niemand beantworten kann: Seit wann steht der Timeout auf diesem Wert? Wer hat am Freitag den Real Server entfernt? Wann läuft welches Zertifikat ab? Und weil das Skript nur liest, läuft es mit dem Nur-Lese-Konto und kann nichts kaputt machen.



Skizze: Beispielablauf; Speicherort und Werkzeuge nach Hausstandard

PowerShell

```
# Nächtliche Dokumentation: lesen, sortiert ablegen, Git zeigt die Unterschiede
$ziel = '<Pfad-zum-Repository>\lm01'

Get-AdcVirtualService |
    ConvertTo-Json -Depth 8 |
    Set-Content -Path "$ziel\virtualservices.json" -Encoding utf8

Get-TlsCertificate |
    Select-Object * |
    Export-Csv -Path "$ziel\zertifikate.csv" -NoTypeInformation -Encoding utf8

Set-Location $ziel
git add -A
git commit -m "LoadMaster-Konfiguration $(Get-Date -Format 'yyyy-MM-dd')" --quiet
```

Damit Git nur echte Änderungen zeigt, sollten die Dateien keine Werte enthalten, die sich bei jedem Lauf ändern, etwa Verbindungszähler oder aktuelle Statistiken. Filtere solche Felder vor dem Speichern heraus. Die Dokumentation ersetzt übrigens kein Backup. Sie hilft dir zu verstehen, was war. Wiederherstellen kannst du damit nichts, dafür gibt es die Konfigurationssicherung des Geräts, die sich ebenfalls per API oder per Modul ziehen lässt. Wie du sie automatisch und außerhalb des Geräts ablegst, steht in Firmware-Updates, Backup und Notfallplan: LoadMaster-Betrieb ohne Herzrasen.

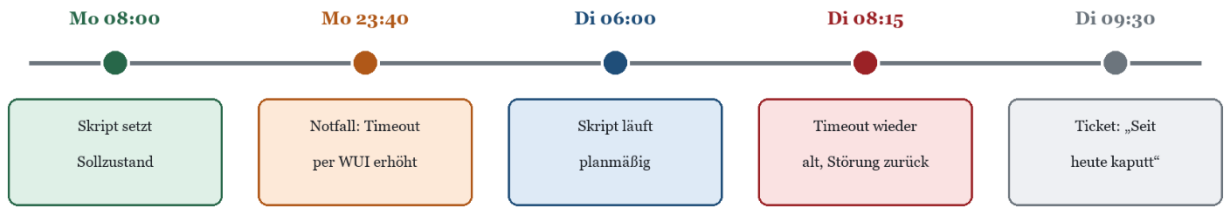
Aufgabe	Konto	Risiko	Wichtigster Schutz
Real Server in Wartung	api-wartung	letzten gesunden Server abschalten, Server bleibt deaktiviert	Vorabprüfung, finally-Block, Monitoring auf deaktivierte Server
Zertifikat austauschen	api-zert	falsches Zertifikat, fehlende Kette, Schlüssel liegt herum	Ersetzen unter gleichem Namen, Test von außen, Datei löschen
Konfiguration dokumentieren	api-doku	Geheimnisse landen im Repository	Nur-Lese-Konto, Felder filtern, Repository mit Zugriffsschutz
Backup ziehen	api-doku oder eigenes Konto	Backup liegt nur auf dem Gerät	Ablage außerhalb, Wiederherstellung üben

Die Falle: Automatisierung, die manuelle Änderungen still überschreibt

Jetzt kommt der Teil, den die Hersteller-Beispiele gern weglassen. Sobald ein Skript nicht nur liest, sondern einen Sollzustand herstellt, gibt es zwei Wahrheiten: die im Skript oder in dessen Konfigurationsdatei und die auf dem Gerät. Solange beide übereinstimmen, ist alles gut. Interessant wird es in der Nacht, in der jemand bei einer Störung schnell per Weboberfläche einen Timeout erhöht, einen Real Server herausnimmt oder eine Content Rule abschaltet. Am nächsten Morgen läuft das Skript, findet eine Abweichung und tut das, wofür es gebaut wurde: Es stellt den alten Zustand wieder her. Ohne Rückfrage, ohne Meldung, mit großer Zuverlässigkeit.

Die stille Rückabwicklung: wenn das Skript die Notfalländerung überschreibt

Montag Nacht klickt jemand eine Korrektur. Dienstag früh stellt das Skript den alten Stand wieder her



Drei Regeln gegen die stille Rückabwicklung

1. Vergleichen, bevor du schreibst: Ist-Wert lesen, mit dem letzten bekannten Wert abgleichen.
2. Abweichung melden statt überbügeln: Unerwartete Werte stoppen das Skript und erzeugen ein Ticket.
3. Jede manuelle Änderung landet in derselben Quelle, aus der das Skript seinen Sollzustand liest.

Skizze: konstruiertes Beispiel; Zeitpunkte und Parameter frei gewählt

Das Tückische daran ist nicht der Fehler selbst, sondern seine Unsichtbarkeit. Die Person, die nachts korrigiert hat, schläft. Das Team, das morgens die Störung bekommt, sieht in der Weboberfläche einen Zustand, der genau so aussieht wie vor einer Woche. Die Spur führt erst dann zum Skript, wenn jemand die Protokolle nebeneinanderlegt. Wer Desired State Configuration oder Gruppenrichtlinien kennt, kennt das Muster: Die Richtlinie gewinnt immer, auch gegen die gut gemeinte Ausnahme.

Erst lesen, dann vergleichen, dann schreiben

Die Abhilfe ist kein Verzicht auf Automatisierung, sondern ein anderes Verhalten bei Abweichungen. Ein Skript, das einen Sollzustand pflegt, liest zuerst den Ist-Zustand, vergleicht ihn mit dem Zustand, den es beim letzten Lauf selbst gesetzt hat, und unterscheidet drei Fälle. Stimmt alles, tut es nichts. Weicht der Ist-Zustand vom letzten eigenen Stand ab, hat jemand von Hand eingegriffen, und dann ändert das Skript nichts, sondern meldet. Nur wenn der Ist-Zustand dem letzten eigenen Stand entspricht und der Sollzustand sich geändert hat, schreibt es.

PowerShell

```
# Muster: Abweichung melden statt überbügeln (vereinfacht)
$soll   = Get-Content '<Pfad>\soll.json'   | ConvertFrom-Json
$zuletzt = Get-Content '<Pfad>\zuletzt.json' | ConvertFrom-Json
$ist    = <Ist-Wert per Get-Cmdlet lesen>

if ($ist -ne $zuletzt.Wert) {
    Write-Warning "Manuelle Änderung erkannt: $($zuletzt.Wert) wurde zu $ist. Keine Aktion."
    # Ticket oder Mail an das Team, Skript endet hier
    return
}
if ($ist -ne $soll.Wert) {
    <Set-Cmdlet mit $soll.Wert> -WhatIf     # erst mit -WhatIf, dann scharf schalten
    $soll | ConvertTo-Json | Set-Content '<Pfad>\zuletzt.json'
}
```

WARNUNG · Kein Sollzustand ohne Änderungsprozess

Ein Skript, das den LoadMaster auf einen Sollzustand zurücksetzt, ist nur dann sicher, wenn jede Änderung, auch die nächtliche Notfallkorrektur, nachträglich in die Quelle des Sollzustands eingetragen wird. Wenn dein Team diesen Prozess nicht lebt, lass das Skript Abweichungen melden und niemals selbst korrigieren. Ein Melder, der zu oft meldet, nervt. Ein Korrektor, der falsch korrigiert, verursacht Störungen.

Weitere Schutzmaßnahmen für den Betrieb

- **Protokollieren:** Jedes schreibende Skript schreibt per Start-Transcript oder eigenem Protokoll mit, was es geändert hat, und zwar vorher und nachher. Der LoadMaster selbst protokolliert API-Anmeldungen im Syslog.
- **Sperren:** Zwei Skripte, die gleichzeitig am selben LoadMaster arbeiten, sind eins zu viel. Eine einfache Sperrdatei auf dem Automatisierungsserver verhindert parallele Läufe.
- **Bremsen:** Eine kurze Pause zwischen Aufrufen hält dich unter der Grenze von 30 Aufrufen in drei Sekunden, ab der der LoadMaster die Verbindung kappt.
- **Trockenlauf:** Schreibende Funktionen bekommen einen Schalter für einen Trockenlauf, der nur anzeigt, was passieren würde.
- **Ein Ort für Geräte:** Wer mehrere LoadMaster an mehreren Standorten betreibt, sollte prüfen, ob zentrale Verwaltung sinnvoller ist als ein Skript pro Gerät.

Zum letzten Punkt: Kemp 360 Central bündelt Verwaltung, Überwachung und eine eigene API für viele LoadMaster und ist für größere Umgebungen oft die bessere Grundlage als ein Zoo aus

Einzelskripten. Was es leistet und wann es sich lohnt, beschreibt Kemp 360 Central und Global Server Load Balancing für mehrere Standorte.

CHECKLISTE · Bevor ein Skript schreibend an den LoadMaster darf

API nur aus dem Verwaltungsnetz erreichbar, Verwaltungsoberfläche mit Zertifikat aus der eigenen PKI, Zertifikatsprüfung im Skript aktiv.

Eigener API-Benutzer pro Zweck, minimale Rechte, bal nur für Notfälle.

API-Schlüssel statt Kennwort, abgelegt im Tresor, nicht im Skript.

Modulversion festgelegt und dokumentiert, Beispiele mit Get-Help gegen diese Version geprüft.

Vorabprüfung vor jeder Änderung, finally-Block für das Zurückholen, Pause in Schleifen.

Abweichungen vom letzten eigenen Stand werden gemeldet, nicht überschrieben.

Protokoll, Sperre gegen Parallelläufe, Trockenlauf-Schalter.

Monitoring meldet deaktivierte Real Server und bald ablaufende Zertifikate.

Konfigurationsbackup automatisch und außerhalb des Geräts, unabhängig vom Skript.

Wenn du für deine Umgebung ein Rechtekonzept und eine Handvoll belastbarer Skripte brauchst, die auch der Kollege versteht, der sie in zwei Jahren übernimmt, unterstützt dich die Beratung zum Kemp LoadMaster. Wer API und Modul lieber erst gefahrlos ausprobieren möchte, findet das in der Kemp-LoadMaster-Schulung.

FAQ: Kemp LoadMaster per REST-API und PowerShell

Ist die API am LoadMaster standardmäßig aktiv?

Nein. Die API-Schnittstelle ist ab Werk deaktiviert und wird in der Weboberfläche unter Certificates & Security, Remote Access mit Enable API Interface eingeschaltet.

Was ist der Unterschied zwischen API v1 und API v2?

Die API v1 nimmt Befehle als Adresse mit Parametern entgegen und antwortet mit XML. Die API v2 erwartet einen POST mit JSON an /accessv2. Die Zertifikatsanmeldung gibt es nur in v1, einige wenige Befehle ebenfalls.

Welchen Anmeldeweg soll ich für Skripte verwenden?

API-Schlüssel. Progress empfiehlt sie, sie funktionieren mit beiden API-Varianten und lassen sich einzeln widerrufen. Die Zertifikatsanmeldung soll laut Hersteller später abgekündigt werden.

Kann ich den Benutzer bal für Skripte nutzen?

Technisch ja, sinnvoll ist es nicht. bal hat über die API vollen Zugriff, ist im Protokoll nicht einer Aufgabe zuzuordnen und kann keine Zertifikatsanmeldung nutzen. Lege eigene Benutzer an.

Wo bekomme ich das PowerShell-Modul her?

Aus der PowerShell Gallery unter dem Namen Kemp.LoadBalancer.Powershell. Seit Firmware 7.2.36 ist es signiert, je nach Ausführungsrichtlinie musst du das Signaturzertifikat importieren.

Warum bricht mein Skript in einer Schleife plötzlich die Verbindung ab?

Weil der LoadMaster mehr als 30 Aufrufe in weniger als drei Sekunden über alle API-Schnittstellen mit einem Verbindungsabbruch quittiert. Baue eine kurze Pause zwischen die Aufrufe.

Was ist der Unterschied zwischen globalem und VS-bezogenem Deaktivieren eines Real Servers?

Global wirkt in allen Virtual Services, in denen die Adresse als Real Server eingetragen ist. VS-bezogen wirkt nur im angegebenen Dienst. Global ist bequem, erwischt aber auch Dienste, an die du nicht gedacht hast.

Muss ich nach dem Zertifikatstausch die Virtual Services anpassen?

Nicht, wenn du das bestehende Zertifikat unter gleichem Namen ersetzt. Dann bleibt die Zuordnung erhalten. Legst du ein neues Zertifikat unter neuem Namen an, musst du jeden Virtual Service umhängen.

Ersetzt die Dokumentation per Skript das Backup?

Nein. Sie zeigt, was wann geändert wurde. Wiederherstellen kannst du nur aus der Konfigurationssicherung des Geräts, ergänzt um ein Zertifikatsbackup.

Wie verhindere ich, dass ein Skript Notfalländerungen rückgängig macht?

Indem es vor dem Schreiben den Ist-Zustand mit dem zuletzt selbst gesetzten Stand vergleicht und bei einer Abweichung meldet, statt zu korrigieren. Und indem jede manuelle Änderung nachträglich in die Quelle des Sollzustands wandert.

Funktioniert das auch für Exchange-Patchdays mit mehreren Diensten?

Ja, gerade dort lohnt es sich, weil ein Exchange-Server meist in mehreren Virtual Services steckt. Wie diese Dienste am LoadMaster aufgebaut sind, beschreibt Exchange Server hinter dem LoadMaster: Templates, Namespaces, Health Checks.

Fazit: Ein gutes Skript ist ein vorsichtiger Kollege

Die REST-API und das PowerShell-Modul machen den LoadMaster zu einem Gerät, das sich genauso selbstverständlich verwalten lässt wie Exchange oder Windows Server. Real Server für den Patchday herausnehmen, Zertifikate tauschen, jede Nacht die Konfiguration ins Repository schreiben: Das sind Aufgaben, die per Skript schneller, gleichmäßiger und nachvollziehbarer laufen als per Klick. Voraussetzung ist eine Tür, die nur dem Richtigen offensteht: API nur im Verwaltungsnetz, eigene Konten pro Zweck, API-Schlüssel im Tresor.

Der eigentliche Unterschied zwischen einem guten und einem gefährlichen Skript liegt aber nicht in der Syntax. Er liegt darin, ob das Skript vorher fragt, wie die Welt gerade aussieht. Ein Skript, das prüft, bevor es schaltet, das zurückholt, was es weggenommen hat, und das meldet, statt still zu überbügeln, ist ein Kollege, den man gern im Team hat. Eines, das einfach macht, ist der Grund, warum am Dienstagmorgen um Viertel nach acht das Telefon klingelt.

WEITERLESEN · Weiter im Thema

› Firmware-Updates, Backup und Notfallplan: LoadMaster-Betrieb ohne Herzrasen
› Kemp 360 Central und Global Server Load Balancing für mehrere Standorte
› TLS-Offload, Re-Encryption und Zertifikate: PKI trifft LoadMaster
› HA-Paar, Failover und Konfig-Sync: zwei LoadMaster, ein Gedanke
› Monitoring, Syslog und SNMP: den LoadMaster ins Monitoring holen
› Virtual Service, Real Server, SubVS: das Grundvokabular des LoadMasters