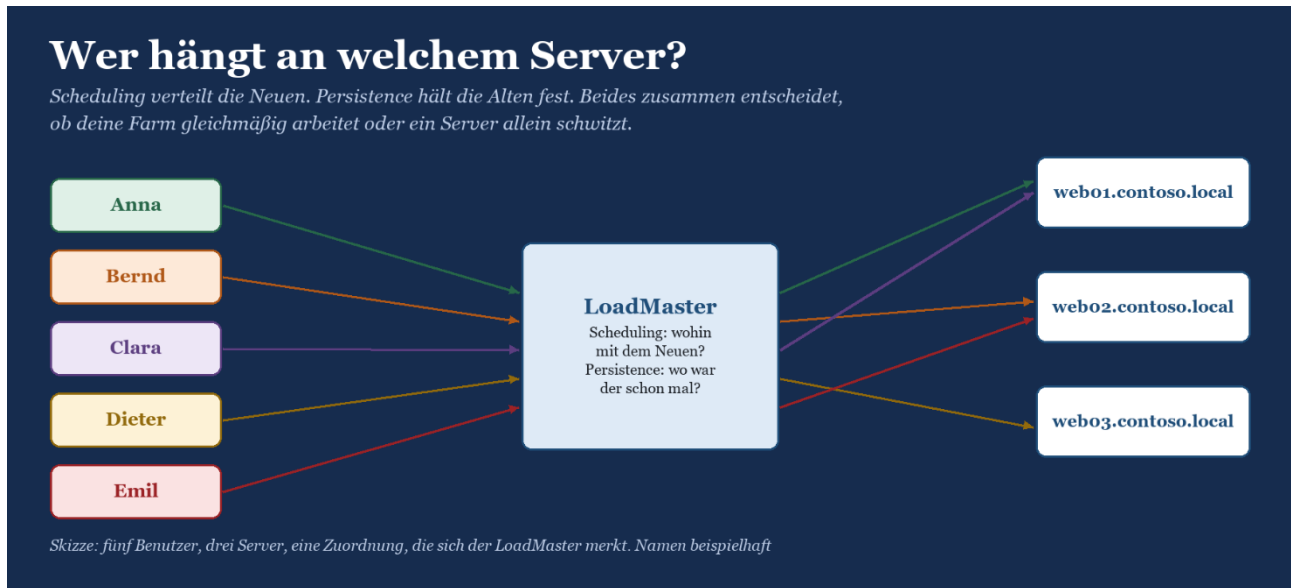


Persistence und Scheduling: wer hängt an welchem Server und warum



WISSEN

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

› [Kemp LoadMaster](#)

BERATUNG

Ein Server glüht, die anderen langweilen sich? Wir schauen gemeinsam auf Scheduling, Persistence und Timeouts deiner Virtual Services.

› [Beratung zum Kemp LoadMaster](#)

SCHULUNG

Grundlagen und Praxis für dein Admin-Team: vom ersten Virtual Service bis zur Persistence, die weiß, was sie tut.

› [Kemp-LoadMaster-Schulung](#)

Jeder Load Balancer beantwortet bei jeder neuen Verbindung zwei Fragen. Erstens: Kenne ich den schon? Zweitens, falls nicht: Wohin mit ihm? Die erste Frage ist Persistence, die zweite ist Scheduling. Klingt banal, und genau deshalb werden beide Einstellungen beim Anlegen eines Virtual Service gern auf dem Wert gelassen, den das Template oder die Kollegin von 2017 gewählt hat. Bis an einem Dienstagvormittag ein Server bei 100 Prozent CPU hechelt, während seine beiden Geschwister im Leerlauf Solitär spielen.

Das Tückische: Beide Einstellungen wirken unsichtbar. Ein falscher Health Check fällt irgendwann auf, ein falsches Zertifikat sofort. Eine unpassende kemp persistence dagegen sieht aus wie Pech, wie „die Anwendung ist halt langsam“ oder wie ein Benutzer, der sich „ständig neu anmelden muss“. Und Microsoft-Workloads machen es nicht leichter: Exchange braucht seit 2013 gar keine Persistence mehr, das RD Gateway dagegen bricht ohne sie zusammen, und SharePoint liegt irgendwo dazwischen, je nachdem, wie deine Benutzer sich anmelden.

In diesem Beitrag sortieren wir die Scheduling-Methoden des LoadMasters von Round Robin bis Adaptive, gehen die Persistence-Modi von Source IP bis SSL Session ID durch und klären, welcher Timeout zu welcher Anwendung passt. Danach gibt es die Microsoft-Brille: Exchange, SharePoint, RD Gateway, RD Web und Eigenentwicklungen. Und zum Schluss die Praxisfalle, die in fast jedem mittelständischen Umfeld irgendwann zuschlägt: Hunderte Benutzer hinter einem NAT-Gateway, eine einzige Quell-IP. Wie sich das Thema in die ganze Serie einordnet, von den Grundlagen bis zur Kerberos-Delegation, zeigt dir der [Kemp LoadMaster: der Wissensbereich](#).

FAKTEN · Scheduling und Persistence in Kürze

Scheduling entscheidet, auf welchem Real Server eine neue Verbindung landet. Der LoadMaster bietet unter anderem Round Robin, Weighted Round Robin, Least Connection, Weighted Least Connection, agentenbasiertes Adaptive Balancing, Fixed Weighting, Weighted Response Time und Source IP Hash. Die Methode gilt je Virtual Service.

Persistence sorgt dafür, dass ein bereits bekannter Client wieder auf demselben Real Server landet. Solange ein gültiger Persistence-Eintrag existiert, wird das Scheduling für diesen Client übergangen. Der Persistence-Timeout bestimmt, wie lange sich der LoadMaster die Zuordnung merkt. Jede neue Verbindung innerhalb des Fensters setzt ihn zurück.

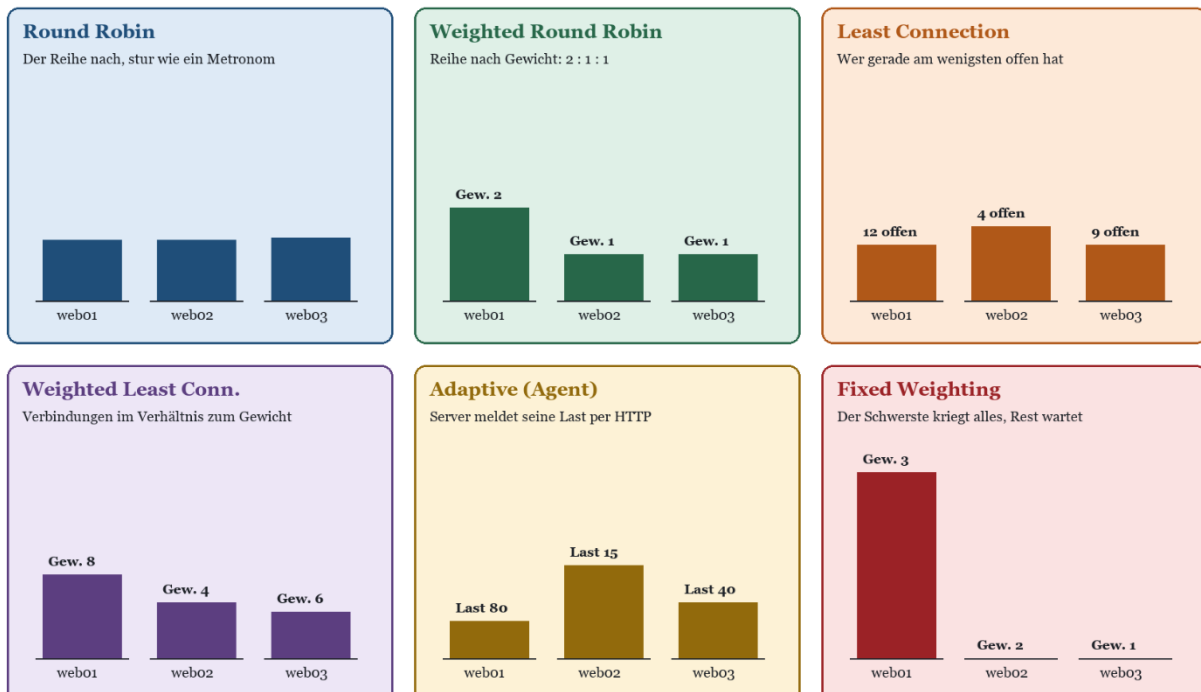
Microsoft schreibt für Exchange 2016, 2019 und Subscription Edition ausdrücklich, dass Session Affinity nicht mehr erforderlich ist, und empfiehlt Layer 7 ohne Affinity mit Health Check je virtuellem Verzeichnis.

Scheduling: wer den nächsten Neuen bekommt

Scheduling ist der Teil, den alle zu verstehen glauben, weil die Namen so selbsterklärend klingen. Round Robin heißt der Reihe nach, Least Connection heißt der mit den wenigsten Verbindungen. Stimmt auch. Die interessanten Details stecken aber darin, was „Verbindung“ für den LoadMaster bedeutet und wann die Methode überhaupt gefragt wird. Wo Virtual Service, Real Server und SubVS im Gefüge stehen, erklärt der Beitrag [Virtual Service, Real Server, SubVS: das Grundvokabular des LoadMasters](#).

Sechs Arten, Arbeit zu verteilen

Was die Scheduling-Methoden des LoadMasters bei neuen Verbindungen tatsächlich tun



Skizze: Anteil neuer Verbindungen je Server, Werte schematisch

Round Robin und Weighted Round Robin

Round Robin verteilt neue Verbindungen strikt reihum: web01, web02, web03, web01 und so weiter. Die Methode schaut weder auf die Last noch darauf, wie lange eine Verbindung dauert. Für gleich starke Server mit kurzen, ähnlich teuren Anfragen ist das ideal und unschlagbar

vorhersehbar. Sobald aber eine Verbindung zwei Stunden lebt und eine andere zwei Sekunden, ist „gleich viele neue Verbindungen“ nicht mehr dasselbe wie „gleich viel Arbeit“.

Weighted Round Robin ergänzt ein statisches Gewicht je Real Server. Ein Server mit doppeltem Gewicht bekommt zwei neue Verbindungen, während ein schwächerer eine bekommt. Das ist die richtige Wahl, wenn deine Farm aus Hardware verschiedener Generationen besteht oder ein Knoten bewusst weniger tragen soll, etwa weil auf ihm noch der Suchcrawler läuft.

Least Connection und Weighted Least Connection

Least Connection schickt jede neue Verbindung an den Real Server mit den wenigsten aktiven Verbindungen. Das passt hervorragend zu langlebigen Verbindungen, etwa MAPI over HTTP, ActiveSync mit seinem Direct Push oder RDP. Die Gewichtete Variante teilt die Verbindungszahl durch das Gewicht und wählt den Server mit dem kleinsten Verhältnis.

Zwei Eigenheiten solltest du kennen. Bei sehr wenig Verkehr bevorzugt Least Connection den ersten Server der Liste, weil bei Gleichstand eben der Erste gewinnt. Im Testlabor mit drei Klicks sieht das aus wie ein Fehler, ist aber keiner. Und wenn ein Server nach einem Neustart wieder online kommt, hat er null Verbindungen und wäre damit für Least Connection der Lieblingskandidat für alle neuen Verbindungen gleichzeitig. Dafür gibt es eine Slow-Start-Funktion, die die Zahl der Verbindungen zu einem frisch zurückgekehrten Server zunächst begrenzt und dann schrittweise erhöht. Bei Anwendungen mit teurem Kaltstart, also praktisch allem auf IIS mit .NET, ist das Gold wert.

WARNUNG · Der Ansturm auf den Frischling

Ein SharePoint-Frontend kommt nach dem Patchday zurück, Least Connection sieht null Verbindungen und schickt ihm schlagartig ein Drittel aller neuen Benutzer. Der App-Pool kompiliert noch, die ersten Anfragen laufen in Timeouts, der Health Check meldet den Server wieder ab, die Verbindungen gehen zurück auf die anderen, der Health Check meldet ihn wieder an. Wiederholen bis zur Mittagspause. Aktiviere Slow Start oder wärme den Server vor dem Wiedereinhängen auf.

Adaptive, Fixed Weighting und die Exoten

Beim agentenbasierten Adaptive Balancing fragt der LoadMaster jeden Real Server regelmäßig per HTTP GET nach einem Lastwert, einer schlichten Zahl, die beschreibt, wie beschäftigt der Server ist. Daraus berechnet er dynamische Gewichte. Klingt elegant, setzt aber voraus, dass auf jedem Server etwas läuft, das diese Zahl ehrlich liefert. Kann der LoadMaster den Wert nicht abholen oder melden alle Server null, fällt er vorübergehend auf Round Robin zurück. Ein Agent, der nach einem Update still und leise nicht mehr antwortet, verwandelt dein Adaptive also unbemerkt in Round Robin. Kein Drama, aber auch nicht das, was du konfiguriert zu haben glaubst.

Fixed Weighting ist eigentlich kein Lastverteilungsverfahren, sondern ein Aktiv-Passiv-Schalter: Der Real Server mit dem höchsten Gewicht bekommt alles, die anderen erst, wenn er ausfällt. Das ist praktisch für Anwendungen, die partout nur auf einem Knoten gleichzeitig laufen dürfen, etwa eine Eigenentwicklung mit lokaler Dateiablage. Weighted Response Time errechnet die Gewichte aus den Antwortzeiten der Health Checks und aktualisiert sie etwa alle 15 Sekunden. Und Source IP Hash ordnet Client-Adressen über eine feste Tabelle mit 4096 Plätzen den Real Servern zu, verteilt nach Gewicht. Das ergibt eine Zuordnung ohne Persistence-Tabelle, mit allen Nachteilen der Quell-IP, zu denen wir gleich kommen.

TIPP · Source IP Hash und krumme Gewichte

Die Hash-Tabelle hat 4096 Plätze, die nach Gewicht auf die Real Server verteilt werden. Das

Standardgewicht ist 1000. Geht die Summe der Gewichte nicht glatt in 4096 auf, bekommen einzelne Server, oft die zuletzt hinzugefügten, auffallend wenig oder gar keine Verbindungen. Wähle die Gewichte so, dass ihre Summe ein Teiler von 4096 ist, zum Beispiel vier Server mit je 1024.

Welche Methode für welchen Dienst

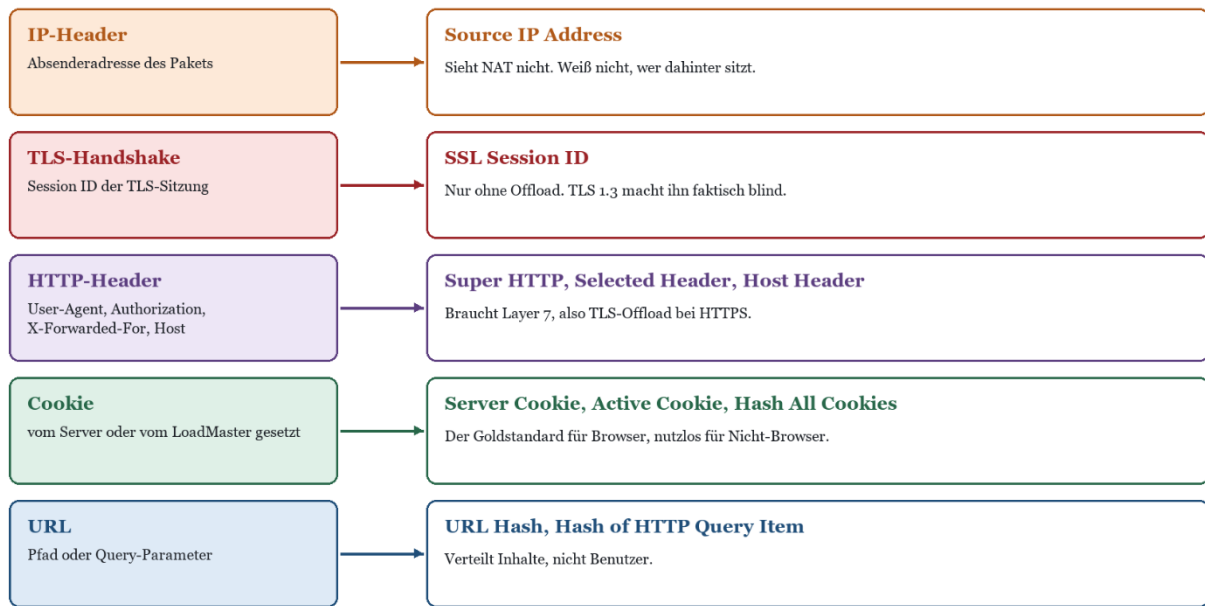
Methode	Wie sie entscheidet	Stärke	Schwäche	Typischer Einsatz
Round Robin	Strikt reihum	Vorhersehbar, kein Overhead	Ignoriert Last und Verbindungsdauer	Gleich starke Server, kurze Anfragen
Weighted Round Robin	Reihum nach statischem Gewicht	Ungleiche Hardware abbildbar	Gewicht passt sich nie an	Gemischte Hardware, Knoten mit Nebenaufgaben
Least Connection	Wenigste aktive Verbindungen	Gut bei langlebigen Verbindungen	Ansturm auf zurückkehrende Server ohne Slow Start	Exchange, RD Gateway, RDP
Weighted Least Connection	Verbindungen geteilt durch Gewicht	Kombiniert beides	Gewichte wollen gepflegt werden	Gemischte Farm mit langen Sitzungen
Adaptive (Agent)	Lastwert, den der Server meldet	Echte Last statt Annahmen	Agent nötig, fällt still auf Round Robin zurück	Rechenintensive Eigenentwicklungen
Fixed Weighting	Höchstes Gewicht bekommt alles	Saubere Aktiv-Passiv-Logik	Keine Lastverteilung	Anwendungen, die nur einmal laufen dürfen
Source IP Hash	Feste Hash-Tabelle über die Quell-IP	Zuordnung ohne Tabelle	NAT, Gewichtsarithmetik	Layer-4-Dienste ohne andere Merkmale

Persistence: das Gedächtnis des LoadMasters

Persistence ist im Kern eine Tabelle: Merkmal des Clients, zugeordneter Real Server, Ablaufzeit. Kommt eine neue Verbindung, schaut der LoadMaster nach, ob er das Merkmal kennt. Wenn ja, geht die Verbindung dorthin, Scheduling hin oder her. Wenn nein, entscheidet das Scheduling, und das Ergebnis wird eingetragen. Die ganze Kunst liegt in der Frage, welches Merkmal einen Benutzer zuverlässig wiedererkennt, ohne versehentlich hundert Benutzer in einen Topf zu werfen.

Woran erkennt der LoadMaster einen Wiederkehrer?

Jeder Persistence-Modus liest ein anderes Merkmal aus Verbindung oder Anfrage



Skizze: Merkmalsquellen der Persistence-Modi, vereinfacht

Source IP: simpel, robust, naiv

Source IP Address ist der älteste und einfachste Modus. Er funktioniert für jedes TCP-Protokoll, auch für Dinge, die mit HTTP nichts am Hut haben, und er braucht kein TLS-Offload. Seine Schwäche ist gleichzeitig sein Prinzip: Er erkennt Netzwerkadressen, keine Menschen. Sitzen viele Benutzer hinter einer Adresse, landen sie alle auf einem Server. Wechselt ein Benutzer die Adresse, etwa ein Smartphone vom WLAN ins Mobilfunknetz, verliert er seine Zuordnung. Beides ist in der Praxis häufiger, als dir lieb ist.

Cookies: Server Cookie, Active Cookie und Hash All Cookies

Für Browser-Anwendungen sind Cookies das treffsicherste Merkmal, weil sie am Browser hängen und nicht an der Netzwerkadresse. Beim Modus Server Cookie liest der LoadMaster einen Cookie, den die Anwendung selbst setzt. Du musst ihm dafür den Namen des Cookies nennen, und der Wert sollte je Benutzer eindeutig sein. Ein Sitzungscookie von ASP.NET erfüllt das, ein Sprach-Cookie mit dem Wert „de-DE“ eher nicht. Wer den falschen Cookie wählt, baut sich eine Persistence nach Landessprache.

Beim Modus Active Cookie setzt der LoadMaster selbst einen Cookie per Set-Cookie, wenn der Client noch keinen mitbringt, und erkennt den Client bei jeder weiteren Anfrage daran. Bestehende Cookies der Anwendung bleiben unangetastet. Das ist für die meisten Webanwendungen der bequemste Weg, weil am Server nichts zu konfigurieren ist. Die Varianten „Server Cookie or Source IP“ und „Active Cookie or Source IP“ fallen auf die Quell-IP zurück, wenn der Cookie fehlt, also typischerweise bei der allerersten Anfrage oder bei Clients, die keine Cookies annehmen. Hash All Cookies bildet einen Hash über alle Cookie-Werte. Ändert die Anwendung unterwegs einen Cookie, und das tun Anwendungen mit Anmeldung praktisch immer, ist der Benutzer für den LoadMaster plötzlich ein Fremder.

WICHTIG • Cookies brauchen Layer 7

Jeder cookie- oder headerbasierte Modus setzt voraus, dass der LoadMaster in die HTTP-Anfrage hineinsehen kann. Bei HTTPS heißt das: TLS endet am LoadMaster, entweder als reines Offload oder

mit Re-Encryption zum Real Server. Ohne Offload sieht er nur verschlüsselte Bytes und kann dir mit dem schönsten Cookie-Modus nicht helfen.

› [TLS-Offload, Re-Encryption und Zertifikate: PKI trifft LoadMaster](#)

Super HTTP, Selected Header und der Rest

Super HTTP bildet einen Fingerabdruck aus dem User-Agent und, falls vorhanden, dem Authorization-Header. Gleicher Fingerabdruck, gleicher Server. Das war eine clevere Idee für Clients, die keine Cookies können, und Kemp hat den Modus lange für bestimmte Webdienste von Lync und Skype for Business empfohlen. Im Unternehmensnetz hat er aber eine bittere Pointe: Wenn alle Arbeitsplätze per Softwareverteilung dieselbe Edge-Version haben, schicken sie alle denselben User-Agent. Ohne Authorization-Header ist die halbe Belegschaft dann ein einziger Fingerabdruck.

Selected Header nimmt den Wert eines frei wählbaren Headers als Merkmal, etwa X-Forwarded-For, das ein vorgelagerter Proxy mit der echten Client-Adresse füllt. Das ist die Rettung, wenn ein Reverse Proxy oder eine Firewall vor dem LoadMaster alle Verbindungen mit der eigenen Adresse anliefert. URL Hash, HTTP Host Header und Hash of HTTP Query Item verteilen dagegen nach Inhalt statt nach Benutzer. Das ist für Caches und Content-Server interessant, für Microsoft-Workloads mit Benutzersitzungen praktisch nie. Für RDP gibt es eigene Modi, die die Benutzererkennung im RDP-Cookie der Verbindungsanfrage auswerten.

SSL Session ID: der Modus, den du vergessen darfst

SSL Session ID ordnet Clients nach der Kennung ihrer TLS-Sitzung zu. Das klingt für HTTPS ohne Offload verlockend, scheitert aber in der Praxis an den Clients. Browser handeln Sitzungen neu aus, wann es ihnen passt, und der Hersteller selbst rät für HTTPS davon ab. Am LoadMaster steht der Modus nur bei Service Type „Generic“ ohne SSL-Beschleunigung zur Verfügung. Und mit TLS 1.3 ist der Gedanke endgültig erledigt: Dort wird die Sitzungswiederaufnahme über Pre-Shared Keys und Tickets erledigt, das Feld für die Session ID existiert nur noch aus Kompatibilitätsgründen und enthält bei vielen Clients bei jedem Handshake einen neuen Zufallswert. Persistence nach Würfelergebnis, sozusagen.

Persistence-Modi im Vergleich

Modus	Merkmal	Braucht Layer 7	Stolperstein	Microsoft-Einsatz
Source IP Address	Quell-IP des Pakets	Nein	NAT, wechselnde Adressen	RD Gateway, nicht-HTTP-Dienste
Server Cookie	Cookie der Anwendung	Ja	Cookie muss je Benutzer eindeutig sein	Eigenentwicklungen mit Sitzungscookie
Active Cookie	Vom LoadMaster gesetzter Cookie	Ja	Nur für Browser-Clients	RD Web, SharePoint mit Formular- oder SAML-Anmeldung
... or Source IP	Cookie, sonst Quell-IP	Ja	Fallback erbt die NAT-Schwäche	Gemischte Clients
Super HTTP	User-Agent plus Authorization	Ja	Einheitliche Browser, gleicher Abdruck	Altlasten, kaum noch sinnvoll
Selected Header	Wert eines Headers	Ja	Header muss vertrauenswürdig sein	Hinter Proxy mit X-Forwarded-For
SSL Session ID	TLS-Sitzungskennung	Nein	Browser, TLS 1.3	Keiner

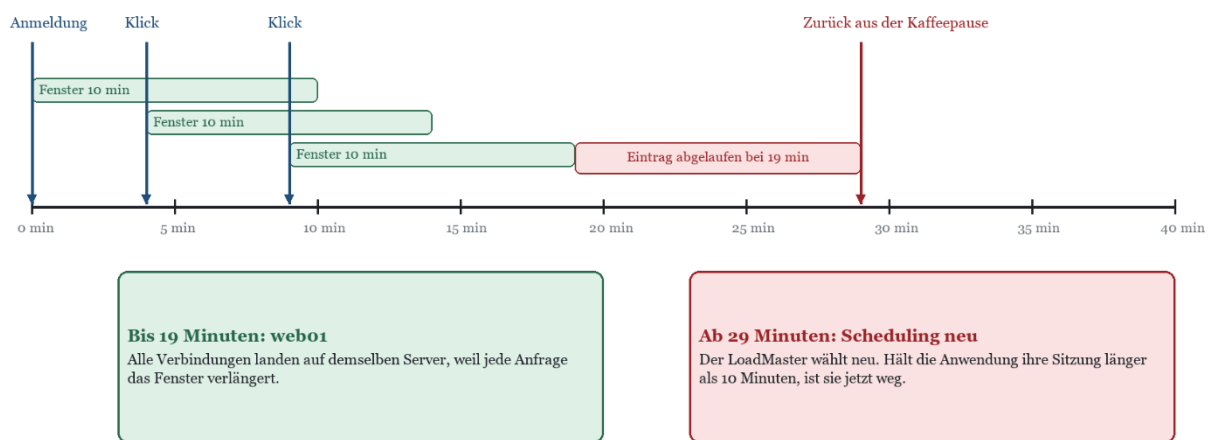
URL, Host, Query	Inhalt der Anfrage	Ja	Verteilt Inhalte, nicht Benutzer	Caches, Content-Server
------------------	--------------------	----	----------------------------------	------------------------

Timeouts: wie lange sich der LoadMaster erinnert

Der Persistence-Timeout ist die am meisten unterschätzte Zahl in der ganzen Konfiguration. Er legt fest, wie lange ein Eintrag nach der letzten Verbindung des Clients gültig bleibt. Jede neue Verbindung innerhalb des Fensters setzt die Uhr zurück. Ist der Timeout zu kurz, verliert ein Benutzer nach der Kaffeepause seinen Server, und mit ihm alles, was die Anwendung lokal im Speicher über ihn wusste. Ist er zu lang, kleben Benutzer noch Stunden an einem Server, den du längst entlasten wolltest, und nach einem Ausfall und der Rückkehr eines Knotens dauert es ewig, bis die Last sich wieder ausgleicht.

Wie der Persistence-Timeout tickt

Jede neue Verbindung innerhalb des Fensters setzt die Uhr zurück. Stille lässt den Eintrag sterben



Skizze: Persistence-Timeout von 10 Minuten, Zeitachse schematisch

Die Faustregel, die auch der Hersteller nennt: Stimme den Timeout auf das Sitzungs-Timeout der Anwendung ab. Hält ASP.NET eine Sitzung 20 Minuten, ergibt ein Persistence-Timeout von fünf Minuten keinen Sinn, einer von acht Stunden aber auch nicht. Ein Puffer von einigen Minuten über dem Anwendungswert ist vernünftig.

Szenario	Sinnvoller Bereich	Begründung
Webanwendung mit InProc-Sitzung	Sitzungs-Timeout der Anwendung plus Puffer	Sonst ist der Warenkorb nach der Pause weg
RD Gateway	Länger als die übliche Sitzungsdauer, Stunden statt Minuten	Lange RDP-Sitzungen mit wenig neuen Verbindungen
RD Web mit Formularanmeldung	Etwas über dem Timeout der Formularanmeldung	Anmeldecookie und Server sollen gemeinsam altern
Dienst ohne Persistence-Bedarf	Kein Timeout, weil keine Persistence	Jede Zuordnung kostet Gleichverteilung

Die Microsoft-Brille: wer Persistence braucht und wer nicht

Die wichtigste Erkenntnis vorweg: Persistence ist kein Qualitätsmerkmal, sondern ein Zugeständnis. Jede Anwendung, die ohne sie auskommt, ist besser verteilbar, übersteht den Ausfall eines Knotens eleganter und lässt sich einfacher warten. Persistence brauchst du genau

dann, wenn eine Anwendung Zustand auf einem einzelnen Server hält, den die anderen nicht kennen.

Exchange ab 2013: Finger weg

Exchange ist das Musterbeispiel für einen Dienst, der Persistence nicht mehr braucht. Seit Exchange 2013 rendert der Server mit der aktiven Datenbankkopie die Sitzung, der angesprochene Server reicht die Anfrage nur durch. Alle Clients sprechen HTTP, und das Anmeldecookie wird mit dem Zertifikat verschlüsselt, das auf allen Servern gleich ist. Jeder Server kann also jedes Cookie lesen. Microsoft empfiehlt für Exchange 2016, 2019 und Subscription Edition ausdrücklich Layer 7 ohne Session Affinity mit Health Check je virtuellem Verzeichnis und weist darauf hin, dass Affinity Kapazität und Auslastung verschlechtert.

AUS DER PRAXIS · Wenn Persistence ein Symptom versteckt

In einer Exchange-Umgebung mussten sich OWA-Benutzer sporadisch neu anmelden. Die schnelle Lösung: Active Cookie am Virtual Service eingeschaltet, Problem weg, alle glücklich. Ein halbes Jahr später fiel bei einem Server-Neustart auf, dass dieser Server seit der letzten Zertifikatserneuerung ein anderes Zertifikat hatte als seine Kollegen. Die Anmeldecookies ließen sich deshalb nicht serverübergreifend entschlüsseln. Persistence hatte den Fehler nicht behoben, sondern nur dafür gesorgt, dass ihn niemand mehr sah. Nach dem Zertifikatstausch flog die Persistence wieder raus.

Wie du Exchange am LoadMaster mit Templates, Namespaces und den passenden Health Checks aufsetzt, steht in [Exchange Server hinter dem LoadMaster: Templates, Namespaces, Health Checks](#). Für langlebige Verbindungen wie MAPI over HTTP und ActiveSync ist dort Least Connection eine gute Wahl.

SharePoint: kommt auf die Anmeldung an

Seit SharePoint 2013 liegen die Anmeldetokens der Benutzer im Logon Token Cache des Distributed Cache, also farmweit. Ein Benutzer mit gültigem FedAuth-Cookie kann deshalb grundsätzlich von jedem Frontend bedient werden. In der Subscription Edition ist der Distributed Cache fest in SharePoint eingebaut, AppFabric als separates Produkt ist Geschichte. Auf dem Papier ist SharePoint damit persistence-tauglich ohne Persistence. In der Praxis gibt es drei Situationen, in denen du sie trotzdem willst:

- **NTLM:** NTLM authentifiziert nicht eine Anfrage, sondern eine TCP-Verbindung. Der dreistufige Handshake muss auf derselben Verbindung zum selben Server stattfinden. Solange der LoadMaster eine Client-Verbindung auch serverseitig auf einer Verbindung hält, geht das gut. Sobald irgendwo Verbindungen gebündelt oder einzelne Anfragen neu verteilt werden, regnet es 401-Fehler.
- **SAML und WS-Federation:** Während der Anmeldung wandert der Browser zwischen SharePoint und Identitätsanbieter hin und her. Ist der Distributed Cache nicht gesund, etwa nach einem missglückten Neustart eines Cache-Hosts, erleben Benutzer ohne Persistence Anmeldeschleifen. Active Cookie ist hier eine verbreitete Absicherung, ersetzt aber keinen funktionierenden Cache.
- **Lösungen mit lokalem Zustand:** Ältere Farmlösungen, Webparts mit InProc-Sitzung oder Dienste von Drittanbietern halten mitunter Daten im Arbeitsspeicher eines Frontends. Die interessiert der Distributed Cache nicht.

Welcher Weg zu welcher Anmeldung passt und wie du die Farm am LoadMaster aufsetzt, beschreibt der Beitrag [SharePoint Server Subscription Edition hinter dem LoadMaster](#). Warum NTLM hinter dem Load Balancer überhaupt so oft auftaucht, obwohl alle Kerberos wollten, erklärt

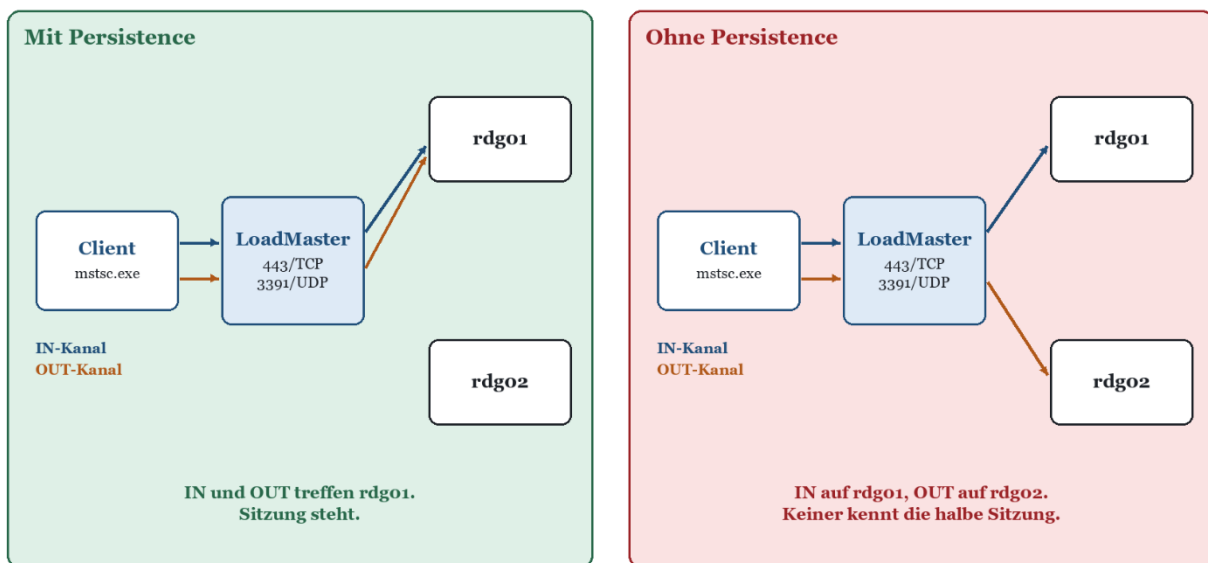
[NTLM, NTLM-Proxy und der Kerberos-Fallback, den keiner bemerkt](#). Den Gesamtüberblick über SharePoint findest du im [SharePoint-Wissensbereich](#).

Remote Desktop Services: Gateway, Web und Broker

Beim RD Gateway ist Persistence keine Option, sondern Pflicht. Der HTTP-Transport öffnet je Sitzung zwei Kanäle, einen für eingehende und einen für ausgehende Daten. Beide müssen auf demselben Gateway landen, sonst hält jeder Server eine halbe Sitzung und keiner kann etwas damit anfangen. Microsoft hat das schon für DNS Round Robin ausgeschlossen und für den Azure Load Balancer extra Client-IP-Affinity nachgerüstet. Am LoadMaster ist Source IP hier das Mittel der Wahl, weil der RDP-Client keine Cookies spielt. Kommt der UDP-Transport auf Port 3391 dazu, muss auch dieser zum selben Gateway wie der HTTPS-Teil. Dafür gibt es am LoadMaster eine Funktion, die Persistence über Virtual Services mit verschiedenen Ports hinweg teilt.

RD Gateway: zwei Kanäle, ein Ziel

Eingangs- und Ausgangskanal einer Sitzung müssen auf demselben Gateway landen



Skizze: HTTP-Transport des RD Gateway mit und ohne Persistence, Namen beispielhaft

RD Web Access ist eine ASP.NET-Anwendung mit Formularanmeldung. Das Anmeldecookie ist mit dem machineKey des Servers geschützt. Haben die RD-Web-Server unterschiedliche Schlüssel, und das ist der Auslieferungszustand, kann Server B das Cookie von Server A nicht lesen, und der Benutzer steht wieder an der Anmeldemaske. Du hast zwei Wege: einen einheitlichen machineKey auf allen Servern oder Active Cookie am LoadMaster. Sauberer ist der einheitliche Schlüssel, robuster ist beides. Für den Connection Broker und die Session Hosts kommt die Verbindung über Redirection und Routing-Token ohnehin beim richtigen Host an; Details zu dieser Kette stehen in [Remote Desktop Services: RD Gateway, RD Web und Connection Broker balancieren](#).

Eigenentwicklungen: erst fragen, dann kleben

Bei Eigenentwicklungen auf IIS hilft keine Herstellerdoku, sondern nur ein Gespräch mit dem Entwicklungsteam. Die entscheidende Frage lautet: Wo liegt der Sitzungszustand? Liegt er InProc im Arbeitsspeicher des Worker-Prozesses, brauchst du Persistence, und zwar mit einem Timeout, der zum Sitzungs-Timeout passt. Liegt er im SQL Server, einem State Server oder einem verteilten Cache, brauchst du keine. Bei ASP.NET-Formularanmeldung oder Data Protection gilt außerdem dasselbe wie bei RD Web: Schlüssel auf allen Servern synchronisieren, sonst nützt die schönste Persistence nach einem Failover nichts. Mehr dazu in [IIS-Eigenentwicklungen und http.sys-Dienste veröffentlichen](#).

TIPP · Der Rapid-Fail-Test

Ob eine Anwendung wirklich ohne Persistence auskommt, findest du in fünf Minuten heraus: Melde dich an, arbeite kurz, und deaktiviere dann den Real Server, auf dem du gelandet bist, am LoadMaster. Wenn du ohne erneute Anmeldung und ohne Datenverlust weiterarbeiten kannst, brauchst du keine Persistence. Wenn nicht, weißt du jetzt, was bei jedem ungeplanten Ausfall passiert, mit oder ohne Persistence.

Persistence-Bedarf der Microsoft-Workloads

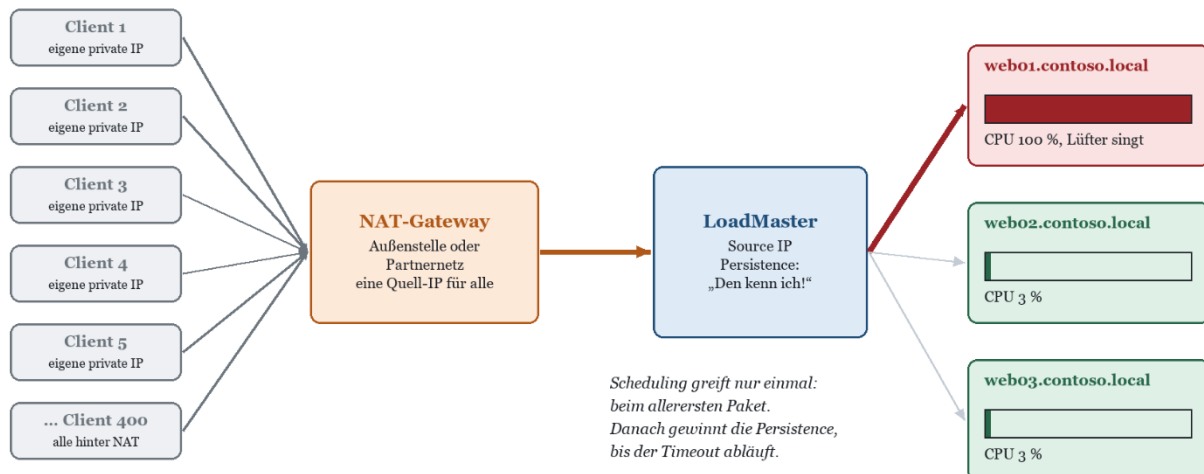
Workload	Persistence nötig?	Empfehlung	Warum
Exchange 2016, 2019, SE	Nein	Keine, Least Connection oder Round Robin	Zustandslose Client-Zugriffsdienste, Cookie mit gemeinsamem Zertifikat
SharePoint mit Kerberos	In der Regel nein	Keine oder Active Cookie als Sicherheitsnetz	Logon Token Cache im Distributed Cache
SharePoint mit NTLM	Verbindungsstreuung nötig	Keine Bündelung, notfalls Source IP oder Active Cookie	NTLM authentifiziert die TCP-Verbindung
SharePoint mit SAML	Empfehlenswert	Active Cookie	Anmeldeschleifen bei Cache-Problemen vermeiden
ADFS	In der Regel nein	Keine	Farm teilt Konfiguration und Schlüssel
RD Gateway	Ja, zwingend	Source IP, UDP an denselben Server	IN- und OUT-Kanal müssen zusammenbleiben
RD Web Access	Ja, außer bei gleichem machineKey	Active Cookie	Formularanmeldung pro Server verschlüsselt
Eigenentwicklung InProc	Ja	Server Cookie oder Active Cookie	Sitzung liegt im Arbeitsspeicher eines Servers

Praxisfalle NAT-Gateway: alle Clients, eine Quell-IP

Jetzt zu dem Fall, der in kaum einem Projekt fehlt. Die Ausgangslage ist harmlos: Ein Virtual Service mit Source-IP-Persistence, drei Real Server, alles läuft. Dann wird eine Außenstelle über einen neuen VPN-Konzentrator angebunden, ein Partnerunternehmen greift über sein Internet-Gateway zu, oder die Verwaltung schickt allen Internetverkehr zentral über eine Firewall mit Source NAT. Ab diesem Moment kommen 400 Menschen mit einer einzigen Quell-IP beim LoadMaster an.

Die NAT-Falle: 400 Menschen, eine IP-Adresse

Für Source-IP-Persistence ist die ganze Außenstelle ein einziger, sehr fleißiger Benutzer



Skizze: Source-IP-Persistence hinter einem NAT-Gateway, Namen beispielhaft

Was dann passiert, ist logisch, aber grausam: Die erste Verbindung aus der Außenstelle wird per Scheduling verteilt, sagen wir auf web01. Ab da kennt der LoadMaster die Adresse. Jede weitere Verbindung aller 400 Benutzer geht auf web01, und weil ständig jemand klickt, läuft der Timeout nie ab. web02 und web03 bedienen nur noch die Handvoll Leute, die direkt im Hauptstandort sitzen. Im Monitoring sieht das aus wie ein defekter Server, denn web01 ist der einzige, der langsam ist. Wer ihn dann neu startet, verschiebt das Problem lediglich auf web02. Dunkle Komik: Je besser dein Timeout auf die Anwendung abgestimmt ist, desto zuverlässiger hält die Zuordnung.

WARNUNG · Auch dein eigenes Netz kann NAT machen

Die Falle kommt nicht nur von außen. Ein vorgelagerter Reverse Proxy, eine Web Application Firewall vor dem LoadMaster oder ein zweiter Load Balancer in der DMZ liefern ebenfalls alle Verbindungen mit der eigenen Adresse an. Wenn ein Virtual Service ausschließlich „einen“ Client sieht, schau nach, wer davor steht.

Symptome erkennen

- **Ungleiche Verteilung trotz Round Robin:** Die Statistik der Real Server zeigt bei gleichen Gewichten dauerhaft eine deutliche Schieflage, die mit dem Scheduling nicht zu erklären ist.
- **Lange Persistence-Liste, wenige Einträge:** In der Übersicht der Persistence-Einträge tauchen auffällig wenige Quelladressen auf, eine davon mit sehr viel Verkehr.
- **Lastprobleme wandern beim Neustart:** Nach einem Neustart ist der vorher überlastete Server ruhig und ein anderer glüht. Klassisches Zeichen für eine große Gruppe, die gemeinsam umzieht.
- **IIS-Logs zeigen eine einzige Client-IP:** Wenn der LoadMaster transparent arbeitet oder X-Forwarded-For protokolliert wird, steht dort für die ganze Außenstelle dieselbe Adresse.

Gegenmittel

Gegenmittel	Wann es passt	Was du bedenken musst
Persistence ganz abschalten	Anwendung kommt ohne aus (Exchange, ADFS, saubere Farmen)	Erst mit dem Rapid-Fail-Test prüfen

Active Cookie statt Source IP	Browser-Anwendungen mit TLS-Offload	Nicht-Browser-Clients brauchen einen Fallback
Selected Header mit X-Forwarded-For	Vorgelagerter Proxy, der die echte Adresse mitschickt	Nur Headern von vertrauenswürdigen Systemen glauben
Kürzerer Timeout	Notlösung, wenn nichts anderes geht	Hilft kaum, solange ständig jemand klickt
Eigener Virtual Service für den Partner	Bekannte, große NAT-Quelle mit eigenem Namen	Mehr Konfiguration, dafür klare Trennung

Für das RD Gateway, das Source IP wirklich braucht, gibt es übrigens keinen eleganten Ausweg. Hier hilft nur ehrliches Sizing: Jedes Gateway muss die größte NAT-Gruppe allein tragen können. Wie du die echte Client-Adresse überhaupt bis zum Server durchreichst und was Transparenz und SNAT dabei bedeuten, zeigt [Client-IP sichtbar machen: Transparenz, SNAT und X-Forwarded-For](#). Wenn du bei einer gewachsenen Konfiguration nicht mehr sicher bist, warum welcher Dienst welche Persistence hat, ist genau das ein typischer Einstieg in die [Beratung zum Kemp LoadMaster](#).

FAKTEN • Was Persistence nicht kann

Persistence hält einen Benutzer nicht auf einem Server, der ausgefallen ist. Meldet der Health Check einen Real Server ab, verteilt der LoadMaster dessen Clients per Scheduling neu, und der Zustand auf dem alten Server ist für die Benutzer verloren. Persistence ersetzt also keine zustandslose Architektur, sie verschiebt nur den Moment, in dem du es merkst.

› [Health Checks, die wirklich prüfen: warum „Port 443 offen“ nicht reicht](#)

FAQ zu Persistence und Scheduling am Kemp LoadMaster

Was ist der Unterschied zwischen Scheduling und Persistence?

Scheduling entscheidet, welcher Real Server eine neue, unbekannte Verbindung bekommt. Persistence sorgt dafür, dass ein bekannter Client wieder beim selben Server landet. Solange ein gültiger Persistence-Eintrag existiert, spielt das Scheduling für diesen Client keine Rolle.

Braucht Exchange Server am LoadMaster Persistence?

Nein. Seit Exchange 2013 sind die Client-Zugriffsdienste zustandslos, und Microsoft empfiehlt für Exchange 2016, 2019 und Subscription Edition Layer 7 ohne Affinity. Wenn sich Benutzer ohne Persistence neu anmelden müssen, liegt meist ein Zertifikatsunterschied zwischen den Servern vor.

Welche Persistence passt zum RD Gateway?

Source IP, weil der RDP-Client keine Cookies unterstützt und IN- und OUT-Kanal derselben Sitzung auf demselben Gateway landen müssen. Nutzt du zusätzlich UDP auf Port 3391, Sorge dafür, dass auch dieser Verkehr zum selben Gateway geht. Wähle den Timeout lang genug für typische Sitzungen.

Warum landen fast alle Benutzer auf einem Server, obwohl Round Robin eingestellt ist?

Fast immer, weil Source-IP-Persistence aktiv ist und viele Benutzer über ein NAT-Gateway oder einen vorgelagerten Proxy mit derselben Adresse ankommen. Das Scheduling greift dann nur ein einziges Mal. Wechsle auf Active Cookie, Selected Header oder schalte Persistence ab, wenn die Anwendung es zulässt.

Kann ich SSL Session ID für HTTPS ohne Offload verwenden?

Technisch ja, sinnvoll ist es nicht. Browser handeln TLS-Sitzungen häufig neu aus, und unter TLS 1.3 ist die Session ID kein verlässliches Merkmal mehr. Für HTTPS ohne Offload bleibt Source IP, mit allen Nachteilen. Besser ist TLS-Offload mit Cookie-Persistence.

Wie lang sollte der Persistence-Timeout sein?

Etwas länger als das Sitzungs-Timeout der Anwendung. Zu kurz kostet Benutzern ihre Sitzung, zu lang verhindert, dass sich die Last nach einem Ausfall wieder gleichmäßig verteilt.

Ist Least Connection immer besser als Round Robin?

Nein. Bei kurzen, gleichartigen Anfragen sind beide praktisch gleichwertig, und Round Robin ist leichter nachzuvollziehen. Least Connection spielt seine Stärke bei langlebigen Verbindungen aus, braucht dann aber Slow Start, damit zurückkehrende Server nicht überrannt werden.

Fazit: so wenig Gedächtnis wie möglich, so viel wie nötig

Scheduling und Persistence sind die zwei Hebel, mit denen du bestimmst, wer an welchem Server hängt. Beim Scheduling ist die Wahl selten dramatisch: Round Robin für kurze Anfragen, Least Connection mit Slow Start für langlebige Verbindungen, Gewichte für ungleiche Hardware. Bei der Persistence dagegen entscheidet sich, ob deine Farm wirklich verteilt oder nur so tut.

Die Microsoft-Welt macht es dir leichter, als viele Altkonfigurationen vermuten lassen. Exchange braucht keine Persistence, ADFS in der Regel auch nicht, und eine sauber gebaute SharePoint-Farm mit Kerberos kommt meist ohne aus. Das RD Gateway braucht sie zwingend, RD Web und Eigenentwicklungen mit lokalem Zustand sehr wahrscheinlich. Und wann immer du Source IP einstellst, frag dich, wie viele Menschen hinter dieser einen Adresse sitzen könnten. Die Antwort ist fast immer: mehr, als dir lieb ist. Wer diese Zusammenhänge mit dem Team am eigenen Labor durchspielen will, findet den Rahmen dafür in der [Kemp-LoadMaster-Schulung](#).

WEITERLESEN • Weiter im Thema

- › [Virtual Service, Real Server, SubVS: das Grundvokabular des LoadMasters](#)
- › [SharePoint Server Subscription Edition hinter dem LoadMaster](#)
- › [Remote Desktop Services: RD Gateway, RD Web und Connection Broker balancieren](#)
- › [Exchange Server hinter dem LoadMaster: Templates, Namespaces, Health Checks](#)
- › [Client-IP sichtbar machen: Transparenz, SNAT und X-Forwarded-For](#)
- › [IIS-Eigenentwicklungen und http.sys-Dienste veröffentlichen](#)