13.7 Load Balancing und Skalierung

Einordnung

Diese Seite gehört zum Kapitel:

Server, Virtualisierung und Hochverfügbarkeit

Auf den vorherigen Seiten ging es um Cluster, Hochverfügbarkeit, Failover, Quorum und Split-Brain.

Jetzt geht es um:

Load Balancing und Skalierung

Diese Themen sind wichtig, wenn Dienste nicht nur ausfallsicherer, sondern auch leistungsfähiger und besser verteilbar betrieben werden sollen.

Grundidee

Ein einzelner Server kann nur eine bestimmte Menge an Anfragen verarbeiten.

Beispiele:

Dienst	Mögliche Last
Webserver	Viele Webseitenaufrufe
API-Server	Viele Anwendungsanfragen
Datenbankserver	Viele Lese- und Schreibzugriffe
Terminalserver	Viele Benutzer gleichzeitig
Fileserver	Viele Dateioperationen
Mailserver	Viele E-Mails und Benutzerzugriffe

Wenn ein Server überlastet ist, gibt es zwei grundsätzliche Möglichkeiten:

Möglichkeit	Bedeutung
Server stärker machen	Mehr CPU, RAM, Storage oder Netzwerk
Mehr Server hinzufügen	Last auf mehrere Systeme verteilen

Was bedeutet Load Balancing?

Load Balancing bedeutet Lastverteilung.

Ein Load Balancer verteilt Anfragen auf mehrere Server.

Beispiel:

Komponente	Aufgabe
Client	Stellt eine Anfrage
Load Balancer	Verteilt die Anfrage
WEB01	Webserver 1
WEB02	Webserver 2
WEB03	Webserver 3

Vereinfacht:

```
Client
  ↓
Load Balancer
  ↓
WEB01 / WEB02 / WEB03
```

Der Benutzer greift auf eine zentrale Adresse zu.
Der Load Balancer entscheidet, welcher Server die Anfrage verarbeitet.

Merksatz

Load Balancing verteilt Anfragen auf mehrere Server, damit Last besser verteilt und Verfügbarkeit erhöht wird.

Warum nutzt man Load Balancing?

Grund	Erklärung
Mehr Leistung	Mehrere Server teilen sich die Arbeit
Höhere Verfügbarkeit	Fällt ein Server aus, können andere übernehmen
Skalierbarkeit	Weitere Server können ergänzt werden
Wartbarkeit	Einzelne Server können aus dem Betrieb genommen werden
Bessere Benutzererfahrung	Anfragen werden schneller beantwortet
Ressourcennutzung	Systeme können gleichmäßiger ausgelastet werden

---

Einfaches Beispiel

Eine Webseite läuft nur auf einem Server.

Problem:

Situation	Folge
Viele Besucher	Server wird langsam
Server fällt aus	Webseite ist nicht erreichbar
Wartung notwendig	Dienst muss abgeschaltet werden

Lösung:

Mehrere Webserver werden betrieben und ein Load Balancer davor geschaltet.

Komponente	Aufgabe
Load Balancer	Nimmt Anfragen entgegen
WEB01	Bearbeitet Teil der Anfragen
WEB02	Bearbeitet Teil der Anfragen
WEB03	Bearbeitet Teil der Anfragen

Load Balancer

Ein Load Balancer ist ein System, das eingehende Anfragen verteilt.

Er kann als Hardware, Software, virtuelle Appliance oder Cloud-Dienst umgesetzt sein.

Beispiele:

Art	Beispiel
Hardware Load Balancer	Spezielle Appliance
Software Load Balancer	HAProxy, Nginx, Apache, Traefik
Cloud Load Balancer	Load Balancer eines Cloud-Anbieters
Firewall mit Load-Balancing-Funktion	Firewall verteilt Anfragen
Kubernetes Service / Ingress	Lastverteilung in Containerumgebungen

---

Backend-Server

Backend-Server sind die Server hinter dem Load Balancer.

Beispiel:

Backend	Aufgabe
WEB01	Webserver
WEB02	Webserver
WEB03	Webserver

Der Client sieht meist nur den Load Balancer.

Die Backend-Server arbeiten im Hintergrund.

Frontend und Backend

Beim Load Balancing unterscheidet man oft zwischen Frontend und Backend.

Begriff	Bedeutung
Frontend	Adresse oder Dienst, den der Client anspricht
Backend	Servergruppe, auf die Anfragen verteilt werden

Beispiel:

Bereich	Beispiel
Frontend	https://intranet.firma.local
Backend	WEB01, WEB02, WEB03

VIP – Virtuelle IP-Adresse

Eine VIP ist eine virtuelle IP-Adresse.

Clients verbinden sich mit dieser IP-Adresse.

Der Load Balancer nimmt die Verbindung an und verteilt sie an Backend-Server.

Beispiel:

Komponente	IP-Adresse
Load Balancer VIP	192.168.10.100
WEB01	192.168.10.101
WEB02	192.168.10.102
WEB03	192.168.10.103

Die Benutzer rufen nur die VIP auf:

```
192.168.10.100
```

Der Load Balancer verteilt dann intern weiter.

DNS und Load Balancing

Oft zeigt ein DNS-Name auf die Adresse des Load Balancers.

Beispiel:

DNS-Name	IP-Adresse
intranet.firma.local	192.168.10.100

Die IP-Adresse gehört zur VIP des Load Balancers.

Vorteil:

Benutzer müssen keine einzelnen Servernamen kennen.

Load Balancing auf verschiedenen Ebenen

Load Balancing kann auf verschiedenen Netzwerkebenen arbeiten.

Ebene	Beispiel	Bedeutung
Layer 4	TCP/UDP	Verteilung nach IP und Port
Layer 7	HTTP/HTTPS	Verteilung nach Inhalt der Anfrage

Layer-4-Load-Balancing

Layer 4 arbeitet auf Transportebene.

Dabei werden Anfragen anhand von IP-Adressen und Ports verteilt.

Beispiel:

Kriterium	Beispiel
Protokoll	TCP
Port	443
Ziel	WEB01 oder WEB02

Der Load Balancer betrachtet nicht den Inhalt der Webseite, sondern nur die Verbindung.

Vorteil:

Schnell und einfach.

Nachteil:

Weniger intelligente Entscheidungen auf Anwendungsebene.

Layer-7-Load-Balancing

Layer 7 arbeitet auf Anwendungsebene.

Bei Webdiensten kann der Load Balancer Inhalte der HTTP/HTTPS-Anfrage auswerten.

Beispiele:

Kriterium	Beispiel
URL-Pfad	/shop geht zu SHOP01
Hostname	api.firma.local geht zu API01
Header	Entscheidung nach HTTP-Header
Cookie	Sitzung einem Server zuordnen

Vorteil:

Sehr flexibel.

Nachteil:

Komplexer und oft rechenintensiver.

Layer 4 vs. Layer 7

Merkmal	Layer 4	Layer 7
Ebene	Transport	Anwendung
Entscheidung nach	IP, Port, Protokoll	URL, Hostname, Header, Cookie
Beispiel	TCP 443 verteilen	/api auf API-Server verteilen
Geschwindigkeit	Oft schneller	Mehr Verarbeitung
Flexibilität	Geringer	Höher

Health Check

Ein Health Check prüft, ob ein Backend-Server erreichbar und funktionsfähig ist.

Beispiel:

Der Load Balancer prüft regelmäßig:

Prüfung	Bedeutung
Ping	Server erreichbar?
TCP-Port	Dienstport offen?
HTTP-Status	Webserver antwortet korrekt?
API-Prüfung	Anwendung funktioniert wirklich?
Datenbankprüfung	Backend kann Datenbank erreichen?

Wenn ein Server nicht gesund ist, nimmt der Load Balancer ihn aus der Verteilung.

Warum Health Checks wichtig sind

Ohne Health Checks könnte der Load Balancer weiterhin Anfragen an einen defekten Server senden.

Beispiel:

Situation	Ohne Health Check	Mit Health Check
WEB02 fällt aus	Benutzer bekommen Fehler	WEB02 wird entfernt
Anwendung hängt	Anfragen landen trotzdem dort	Server wird als fehlerhaft erkannt
Wartung	Benutzer werden gestört	Server kann vorher deaktiviert werden

---

Backend aus der Verteilung nehmen

Ein Backend-Server kann bewusst aus der Lastverteilung genommen werden.

Das nennt man oft:

Begriff	Bedeutung
Disable	Server deaktivieren
Drain	Bestehende Verbindungen auslaufen lassen
Maintenance Mode	Server wird gewartet

Begriff	Bedeutung
Out of Rotation	Server erhält keine neuen Anfragen

Typischer Ablauf bei Wartung:

Schritt	Erklärung
Server aus Rotation nehmen	Keine neuen Anfragen
Bestehende Sitzungen auslaufen lassen	Benutzer nicht hart trennen
Updates installieren	Server warten
Funktion prüfen	Dienst testen
Server wieder aktivieren	Neue Anfragen erlauben

Load-Balancing-Methoden

Der Load Balancer kann unterschiedliche Verfahren nutzen.

Wichtige Methoden:

Methode	Erklärung
Round Robin	Anfragen werden der Reihe nach verteilt
Least Connections	Server mit wenigsten Verbindungen bekommt nächste Anfrage
Weighted Round Robin	Server mit höherem Gewicht bekommt mehr Anfragen
IP Hash	Client-IP entscheidet über Zielserver
Random	Zufällige Verteilung
Response Time	Schnellster Server wird bevorzugt

Round Robin

Round Robin verteilt Anfragen der Reihe nach.

Beispiel:

Anfrage	Zielserver
1	WEB01
2	WEB02
3	WEB03
4	WEB01

Anfrage	Zielserver
5	WEB02

Vorteil:

Einfaches Verfahren.

Nachteil:

Berücksichtigt nicht automatisch, ob ein Server stärker oder schwächer ist.

Least Connections

Least Connections verteilt neue Anfragen an den Server mit den wenigsten aktiven Verbindungen.

Beispiel:

Server	Aktive Verbindungen
WEB01	50
WEB02	20
WEB03	35

Die nächste Anfrage geht an WEB02.

Vorteil:

Sinnvoll, wenn Verbindungen unterschiedlich lange dauern.

Weighted Round Robin

Bei Weighted Round Robin bekommen stärkere Server mehr Anfragen.

Beispiel:

Server	Gewicht
WEB01	3
WEB02	1
WEB03	1

WEB01 bekommt mehr Anfragen, weil er leistungsfähiger ist.

IP Hash

Bei IP Hash wird anhand der Client-IP entschieden, welcher Backend-Server genutzt wird.

Vorteil:

Ein Client landet häufig wieder auf demselben Server.

Nachteil:

Die Verteilung kann ungleichmäßig werden, wenn viele Clients über dieselbe öffentliche IP kommen.

Session Persistence

Session Persistence bedeutet, dass ein Benutzer während einer Sitzung immer wieder beim gleichen Backend-Server landet.

Andere Begriffe:

Begriff	Bedeutung
Sticky Session	Benutzer bleibt am gleichen Server
Session Affinity	Sitzung wird einem Backend zugeordnet

Das ist wichtig, wenn eine Anwendung Sitzungsdaten lokal auf dem Backend-Server speichert.

Warum Sticky Sessions nötig sein können

Beispiel:

Ein Webshop speichert den Warenkorb lokal auf WEB01.

Wenn der Benutzer bei der nächsten Anfrage auf WEB02 landet, kennt WEB02 den Warenkorb nicht.

Lösung:

Lösung	Erklärung
Sticky Session	Benutzer bleibt auf WEB01
Gemeinsamer Session-Speicher	Alle Webserver greifen auf dieselbe Session zu
Anwendung stateless bauen	Keine lokale Sitzung auf Webserver

Nachteil von Sticky Sessions

Sticky Sessions können die Lastverteilung verschlechtern.

Beispiel:

Wenn viele aktive Benutzer auf WEB01 gebunden sind, wird WEB01 stärker belastet als andere Server.

Besser ist oft:

Ansatz	Vorteil
Zentraler Session-Speicher	Benutzer können auf jedem Webserver landen
Stateless-Anwendung	Einfacher zu skalieren
Datenbank oder Redis für Sessions	Einheitlicher Sitzungszustand

---

Stateless

Stateless bedeutet zustandslos.

Ein Server speichert keinen wichtigen Sitzungszustand lokal.

Beispiel:

Jede Anfrage enthält alle nötigen Informationen oder nutzt einen zentralen Speicher.

Vorteil:

Vorteil	Erklärung
Einfacher zu skalieren	Anfragen können auf jeden Server
Einfacher ausfallsicher	Ausfall eines Servers weniger kritisch
Besser für Load Balancing	Keine feste Bindung nötig

Stateful

Stateful bedeutet zustandsbehaftet.

Ein Server speichert Informationen über den aktuellen Zustand einer Sitzung oder Anwendung.

Beispiele:

Zustand	Beispiel

Benutzersitzung	Login-Session
Warenkorb	Webshop
Lokale Datei	Upload liegt nur auf einem Server
Datenbankzustand	Transaktionen

Stateful-Anwendungen sind schwieriger zu skalieren.

Stateless vs. Stateful

Merkmal	Stateless	Stateful
Sitzungsdaten lokal	Nein	Ja
Skalierung	Einfacher	Schwieriger
Load Balancing	Einfacher	Oft Sticky Sessions nötig
Ausfallsicherheit	Besser	Komplexer
Beispiel	Statische Webseite	Warenkorb lokal auf Webserver

Failover vs. Load Balancing

Failover und Load Balancing werden oft verwechselt.

Begriff	Hauptziel
Failover	Übernahme bei Ausfall
Load Balancing	Verteilung von Last

Beispiel Failover:

Eine VM läuft auf HV01.
HV01 fällt aus.
Die VM wird auf HV02 gestartet.

Beispiel Load Balancing:

Drei Webserver laufen gleichzeitig.
Ein Load Balancer verteilt Benutzeranfragen auf alle drei Server.

Failover-Cluster vs. Load-Balancing-Cluster

Merkmal	Failover-Cluster	Load-Balancing-Cluster
Ziel	Ausfallsicherheit	Lastverteilung

Merkmal	Failover-Cluster	Load-Balancing-Cluster
Normalbetrieb	Oft eine aktive Ressource	Mehrere aktive Server
Ausfall	Dienst wechselt oder startet neu	Anfragen gehen an andere Server
Beispiel	VM-Failover	Webserver-Farm
Unterbrechung	Häufig kurz möglich	Oft weniger spürbar

Load Balancing und Hochverfügbarkeit

Load Balancing kann auch zur Hochverfügbarkeit beitragen.

Wenn ein Backend-Server ausfällt, kann der Load Balancer ihn aus der Verteilung nehmen.

Beispiel:

Backend	Zustand
WEB01	Online
WEB02	Ausgefallen
WEB03	Online

Der Load Balancer sendet neue Anfragen nur noch an WEB01 und WEB03.

Wichtig:

Der Load Balancer selbst darf kein Single Point of Failure sein.

Load Balancer als Single Point of Failure

Wenn es nur einen Load Balancer gibt, kann dieser selbst zur Schwachstelle werden.

Beispiel:

Komponente	Risiko
Ein Load Balancer	Fällt er aus, ist der Dienst nicht erreichbar
Mehrere Webserver	Helfen nicht, wenn der Load Balancer ausfällt

Lösung:

Load Balancer redundant betreiben.

Redundante Load Balancer

Bei redundanten Load Balancern gibt es mindestens zwei Load-Balancer-Systeme.

Beispiel:

Load Balancer	Zustand
LB01	Aktiv
LB02	Standby

Fällt LB01 aus, übernimmt LB02.

Das kann über eine virtuelle IP-Adresse erfolgen.

VRRP

VRRP steht für Virtual Router Redundancy Protocol.

Es wird verwendet, um eine virtuelle IP-Adresse zwischen mehreren Systemen hochverfügbar bereitzustellen.

Beispiel:

System	Zustand
LB01	Master
LB02	Backup
VIP	192.168.10.100

Wenn LB01 ausfällt, übernimmt LB02 die VIP.

Keepalived

Keepalived ist eine Software, die häufig VRRP unter Linux bereitstellt.

Typischer Einsatz:

Komponente	Aufgabe
HAProxy	Load Balancing
Keepalived	VIP-Failover zwischen Load Balancern

So kann ein Load Balancer selbst hochverfügbar aufgebaut werden.

Reverse Proxy

Ein Reverse Proxy nimmt Anfragen von Clients entgegen und leitet sie an interne Server weiter.

Er kann auch Load Balancing übernehmen.

Beispiele:

Reverse Proxy	Einsatz
Nginx	Webserver, Reverse Proxy, Load Balancer
HAProxy	Load Balancer und Proxy
Traefik	Reverse Proxy für Containerumgebungen
Apache	Webserver und Reverse Proxy
Caddy	Webserver mit automatischer TLS-Verwaltung

Reverse Proxy vs. Load Balancer

Begriff	Bedeutung
Reverse Proxy	Vermittelt Anfragen an interne Server
Load Balancer	Verteilt Anfragen auf mehrere Server

Ein Reverse Proxy kann gleichzeitig ein Load Balancer sein.

Beispiel:

Nginx nimmt HTTPS-Anfragen entgegen und verteilt sie auf WEB01, WEB02 und WEB03.

SSL/TLS-Terminierung

SSL/TLS-Terminierung bedeutet, dass der Load Balancer oder Reverse Proxy die verschlüsselte HTTPS-Verbindung entgegennimmt und entschlüsselt.

Beispiel:

Verbindung	Verschlüsselung
Client → Load Balancer	HTTPS
Load Balancer → Backend	HTTP oder HTTPS

Vorteile:

Vorteil	Erklärung
Zentrale Zertifikatsverwaltung	Zertifikate nur am Load Balancer

Vorteil	Erklärung
Backend entlasten	Verschlüsselung wird zentral verarbeitet
Inhalt auswertbar	Layer-7-Regeln möglich

Nachteil:

Wenn intern HTTP genutzt wird, muss das interne Netzwerk ausreichend geschützt sein.

Ende-zu-Ende-Verschlüsselung

Bei Ende-zu-Ende-Verschlüsselung bleibt die Verbindung auch bis zum Backend verschlüsselt.

Beispiel:

Verbindung	Verschlüsselung
Client → Load Balancer	HTTPS
Load Balancer → Backend	HTTPS

Vorteil:

Auch interner Verkehr ist verschlüsselt.

Nachteil:

Zertifikatsverwaltung und Fehleranalyse können komplexer werden.

Skalierung

Skalierung bedeutet, ein System leistungsfähiger zu machen.

Es gibt zwei Grundarten:

Art	Bedeutung
Vertikale Skalierung	Einzelnes System stärker machen
Horizontale Skalierung	Weitere Systeme hinzufügen

Scale-Up

Scale-Up bedeutet vertikale Skalierung.

Dabei wird ein einzelner Server stärker gemacht.

Beispiele:

Maßnahme	Wirkung
Mehr RAM	Mehr Arbeitsspeicher
Mehr CPU-Kerne	Mehr Rechenleistung
Schnellere SSDs	Bessere Storage-Performance
Schnellere Netzwerkkarte	Mehr Netzwerkdurchsatz
Größere VM	Mehr Ressourcen für Anwendung

Vorteile von Scale-Up

Vorteil	Erklärung
Einfaches Konzept	Ein System wird stärker
Weniger Architekturänderung	Anwendung bleibt meist gleich
Keine Lastverteilung nötig	Ein Server verarbeitet alles
Gut für bestimmte Anwendungen	Besonders bei nicht verteilbaren Anwendungen

---

Nachteile von Scale-Up

Nachteil	Erklärung
Hardwaregrenze	Irgendwann ist maximale Größe erreicht
Teuer	Große Systeme können sehr teuer sein
Single Point of Failure bleibt möglich	Ein Server bleibt kritisch
Wartung schwieriger	Dienst hängt an einem System
Nicht beliebig skalierbar	Begrenzung durch Hardware

Scale-Out

Scale-Out bedeutet horizontale Skalierung.

Dabei werden mehrere Systeme eingesetzt.

Beispiel:

Statt einem Webserver werden drei Webserver betrieben.

Server	Aufgabe

WEB01	Webserver
WEB02	Webserver
WEB03	Webserver

Ein Load Balancer verteilt die Anfragen.

Vorteile von Scale-Out

Vorteil	Erklärung
Gut erweiterbar	Weitere Server können ergänzt werden
Bessere Ausfallsicherheit	Ausfall eines Servers kann abgefangen werden
Lastverteilung möglich	Arbeit wird verteilt
Wartung einfacher	Einzelne Server können aus Rotation genommen werden
Günstigere Standardhardware möglich	Mehrere kleinere Systeme statt ein sehr großes

---

Nachteile von Scale-Out

Nachteil	Erklärung
Anwendung muss geeignet sein	Nicht jede Anwendung kann verteilt arbeiten
Mehr Komplexität	Load Balancer, Sessions, Datenhaltung
Datenkonsistenz beachten	Mehrere Systeme müssen gleiche Daten sehen
Monitoring aufwendiger	Mehr Komponenten
Netzwerk wichtiger	Kommunikation zwischen Systemen

Scale-Up vs. Scale-Out

Merkmal	Scale-Up	Scale-Out
Prinzip	Ein System stärker machen	Mehr Systeme hinzufügen
Beispiel	Mehr RAM in DB01	WEB01, WEB02, WEB03
Komplexität	Oft einfacher	Komplexer
Grenze	Hardwarelimit	Architekturlimit
Ausfallsicherheit	Nicht automatisch besser	Kann besser werden
Typischer Einsatz	Datenbanken, einzelne Anwendungen	Webserver, APIs, Microservices

---

Vertikale Skalierung bei VMs

Bei virtuellen Maschinen ist Scale-Up oft einfach möglich.

Beispiele:

Änderung	Beispiel
Mehr vCPU	2 vCPU auf 4 vCPU
Mehr RAM	4 GB auf 8 GB
Größere vDisk	80 GB auf 160 GB
Schnellere Storage-Klasse	HDD auf SSD/NVMe

Wichtig:

Mehr Ressourcen helfen nur, wenn die Anwendung dadurch wirklich profitiert.

Horizontale Skalierung bei VMs

Bei horizontaler Skalierung werden mehrere VMs mit gleicher oder ähnlicher Aufgabe betrieben.

Beispiel:

VM	Aufgabe
WEB01	Webserver
WEB02	Webserver
WEB03	Webserver

Ein Load Balancer verteilt die Anfragen.

Das funktioniert besonders gut bei Webservern und APIs.

Skalierung bei Datenbanken

Datenbanken sind schwieriger horizontal zu skalieren als Webserver.

Gründe:

Problem	Erklärung
Datenkonsistenz	Alle Daten müssen korrekt bleiben
Schreibzugriffe	Mehrere schreibende Systeme sind komplex
Transaktionen	Vorgänge müssen vollständig und korrekt sein
Replikation	Daten müssen übertragen werden

Problem	Erklärung
Konflikte	Gleichzeitige Änderungen müssen behandelt werden

Datenbank-Replikation

Replikation bedeutet, dass Daten von einer Datenbank auf eine andere übertragen werden.

Typische Varianten:

Variante	Bedeutung
Primary/Replica	Eine Hauptdatenbank, eine oder mehrere Kopien
Read Replica	Kopie für lesende Zugriffe
Multi-Master	Mehrere Systeme können schreiben
Synchrone Replikation	Bestätigung erst nach Schreiben auf mehrere Systeme
Asynchrone Replikation	Übertragung zeitversetzt

Read Replica

Eine Read Replica ist eine Kopie der Datenbank für lesende Anfragen.

Beispiel:

Datenbank	Aufgabe
DB01	Schreiben und Lesen
DB02	Lesen
DB03	Lesen

Vorteil:

Leselast kann verteilt werden.

Nachteil:

Schreibzugriffe laufen weiterhin über die Hauptdatenbank.

Webserver-Skalierung

Webserver sind oft gut horizontal skalierbar.

Beispiel:

Komponente	Aufgabe
LB01	Load Balancer
WEB01	Webserver
WEB02	Webserver
WEB03	Webserver
DB01	Datenbank

Wichtig:

Alle Webserver müssen auf dieselben Inhalte oder dieselbe Anwendung zugreifen können.

Gemeinsame Daten bei mehreren Webservern

Wenn mehrere Webserver genutzt werden, müssen gemeinsame Daten sauber behandelt werden.

Möglichkeiten:

Lösung	Erklärung
Gemeinsames Dateisystem	Alle Webserver greifen auf gleiche Dateien zu
Deployment auf alle Server	Anwendung wird identisch verteilt
Objekt-Speicher	Dateien liegen zentral
Datenbank	Dynamische Daten liegen zentral
Session-Speicher	Sitzungen liegen zentral, z. B. Redis

Rolling Update

Ein Rolling Update bedeutet, dass Systeme nacheinander aktualisiert werden.

Beispiel:

Schritt	Aktion
1	WEB01 aus Load Balancer nehmen
2	WEB01 aktualisieren
3	WEB01 testen
4	WEB01 wieder aktivieren
5	WEB02 aktualisieren
6	WEB03 aktualisieren

Vorteil:

Der Dienst bleibt während des Updates verfügbar.

Blue-Green-Deployment

Blue-Green-Deployment bedeutet, dass es zwei Umgebungen gibt.

Umgebung	Zustand
Blue	Aktive Produktion
Green	Neue Version

Ablauf:

Schritt	Erklärung
1	Neue Version wird in Green bereitgestellt
2	Green wird getestet
3	Load Balancer schaltet auf Green
4	Blue bleibt als Rückfalloption

Vorteil:

Schneller Rückwechsel möglich.

Canary Deployment

Beim Canary Deployment bekommt zuerst nur ein kleiner Teil der Benutzer die neue Version.

Beispiel:

Anteil	Version
95 %	Alte Version
5 %	Neue Version

Wenn keine Probleme auftreten, wird der Anteil erhöht.

Vorteil:

Fehler betreffen zunächst nur wenige Benutzer.

Autoscaling

Autoscaling bedeutet, dass Systeme automatisch hinzugefügt oder entfernt werden.

Beispiel:

Last	Aktion
CPU hoch	Neue Instanz starten
Wenig Last	Instanz entfernen

Autoscaling ist besonders in Cloud- und Containerumgebungen verbreitet.

In klassischen VM-Umgebungen ist es möglich, aber oft aufwendiger.

Container und Skalierung

Container lassen sich oft sehr gut horizontal skalieren.

Beispiel:

Dienst	Instanzen
Web-App	5 Container
API	3 Container
Worker	4 Container

Container-Orchestrierungssysteme wie Kubernetes können Container automatisch verteilen, überwachen und neu starten.

Kubernetes als Beispiel

Kubernetes ist eine Plattform zur Verwaltung von Containern in Clustern.

Typische Funktionen:

Funktion	Erklärung
Scheduling	Container auf Nodes verteilen
Self-Healing	Fehlerhafte Container neu starten
Scaling	Mehr Instanzen starten
Service Discovery	Dienste auffindbar machen
Rolling Updates	Schrittweise Updates
Load Balancing	Anfragen auf Pods verteilen

Für die IHK reicht meistens das Grundverständnis:

Kubernetes verteilt und verwaltet Container auf mehreren Nodes.

Microservices

Microservices sind kleine, getrennte Dienste, die zusammen eine Anwendung bilden.

Beispiel:

Microservice	Aufgabe
User-Service	Benutzerverwaltung
Payment-Service	Zahlung
Product-Service	Produktdaten
Order-Service	Bestellungen
Mail-Service	E-Mails

Vorteil:

Einzelne Dienste können unabhängig skaliert werden.

Nachteil:

Die Gesamtarchitektur wird komplexer.

Monolithische Anwendung

Eine monolithische Anwendung enthält viele Funktionen in einem großen System.

Beispiel:

Eine Anwendung enthält Benutzerverwaltung, Bestellungen, Zahlung und E-Mail-Funktion in einem Paket.

Vorteil	Nachteil
Einfacher zu verstehen	Schwerer einzeln zu skalieren
Weniger verteilte Komponenten	Änderungen können mehr Risiko haben
Einfacheres Deployment	Größere Abhängigkeiten

Monolith vs. Microservices

Merkmal	Monolith	Microservices
Aufbau	Eine große Anwendung	Viele kleine Dienste
Skalierung	Gesamte Anwendung skalieren	Einzelne Dienste skalieren
Komplexität	Anfangs einfacher	Betrieb komplexer
Deployment	Ein Paket	Viele Deployments
Fehlerauswirkung	Fehler kann große Teile betreffen	Fehler kann begrenzt sein

---

Lastspitzen

Lastspitzen sind kurzzeitig hohe Belastungen.

Beispiele:

Situation	Lastspitze
Viele Benutzer melden sich morgens an	Login-System belastet
Sonderangebot im Webshop	Viele Besucher
Backup startet	Storage und Netzwerk belastet
Monatsabschluss	Datenbank stark belastet
Softwareverteilung	Netzwerk stark belastet

Skalierung und Load Balancing helfen, Lastspitzen besser abzufangen.

Bottleneck

Ein Bottleneck ist ein Engpass.

Beispiele:

Engpass	Auswirkung
CPU	Anwendung rechnet langsam
RAM	System lagert aus
Storage	VMs reagieren träge
Netzwerk	Datenübertragung langsam
Datenbank	Webanwendung langsam
einzelner Server	Dienst überlastet

Merksatz:

Ein System ist oft nur so schnell wie sein größter Engpass.

Kapazitätsplanung

Kapazitätsplanung bedeutet, Ressourcen rechtzeitig zu planen.

Wichtige Fragen:

Frage	Bedeutung
Wie viele Benutzer nutzen den Dienst?	Last abschätzen
Wann treten Lastspitzen auf?	Spitzen einplanen
Welche Ressource ist kritisch?	CPU, RAM, Storage, Netzwerk
Wie schnell wächst die Nutzung?	Zukunft planen
Welche Verfügbarkeit wird benötigt?	HA und Redundanz planen
Wie wird skaliert?	Scale-Up oder Scale-Out

Monitoring für Skalierung

Um richtig zu skalieren, braucht man Monitoring.

Wichtige Werte:

Bereich	Messwert
CPU	Auslastung, Load
RAM	Nutzung, Swapping
Storage	IOPS, Latenz, freier Speicher
Netzwerk	Bandbreite, Paketverlust
Webserver	Anfragen pro Sekunde
Anwendung	Antwortzeit, Fehlerquote
Datenbank	Abfragen, Locks, Replikationsverzögerung
Load Balancer	aktive Verbindungen, fehlerhafte Backends

Antwortzeit

Antwortzeit beschreibt, wie lange ein Dienst braucht, um auf eine Anfrage zu reagieren.

Beispiel:

Antwortzeit	Bewertung
100 ms	Sehr schnell
500 ms	Gut
2 Sekunden	Spürbar langsam
10 Sekunden	Kritisch

Hohe Antwortzeiten können durch CPU, RAM, Storage, Netzwerk oder Datenbankprobleme entstehen.

Throughput

Throughput bedeutet Durchsatz.

Je nach Dienst kann das unterschiedlich gemessen werden.

Beispiele:

Dienst	Throughput
Webserver	Anfragen pro Sekunde
Netzwerk	Mbit/s oder Gbit/s
Storage	MB/s oder IOPS
Datenbank	Abfragen pro Sekunde
Backup	GB pro Stunde

Fehlerquote

Die Fehlerquote zeigt, wie viele Anfragen fehlschlagen.

Beispiele:

Fehler	Bedeutung
HTTP 500	Serverfehler
HTTP 502	Bad Gateway
HTTP 503	Dienst nicht verfügbar
Timeout	Antwort dauert zu lange
Verbindungsfehler	Backend nicht erreichbar

Eine steigende Fehlerquote kann auf Überlastung oder Ausfälle hinweisen.

Load Balancer Monitoring

Beim Load Balancer sollten besonders diese Werte überwacht werden:

Wert	Bedeutung
Aktive Verbindungen	Aktuelle Last
Backend-Status	Server online oder offline
Antwortzeiten	Geschwindigkeit der Backends
Fehlercodes	Probleme bei Anwendungen
Verbindungsabbrüche	Netzwerk- oder Serverprobleme
Traffic	Datenmenge
Health-Check-Fehler	Backend-Probleme

Sicherheit beim Load Balancing

Ein Load Balancer oder Reverse Proxy ist oft ein zentraler Einstiegspunkt.

Daher ist Sicherheit wichtig.

Maßnahmen:

Maßnahme	Erklärung
TLS sauber konfigurieren	Sichere HTTPS-Verbindungen
Zugriff beschränken	Nur notwendige Ports öffnen
Backend-Netz schützen	Backends nicht direkt öffentlich machen
Logging aktivieren	Anfragen nachvollziehen
Updates installieren	Sicherheitslücken schließen
Rate Limiting	Schutz vor zu vielen Anfragen
WAF nutzen	Schutz vor Webangriffen
Adminzugriff absichern	Management nicht öffentlich

Rate Limiting

Rate Limiting begrenzt die Anzahl von Anfragen.

Beispiel:

Ein Client darf maximal 100 Anfragen pro Minute stellen.

Ziel:

Ziel	Erklärung
Schutz vor Missbrauch	Zu viele Anfragen blockieren
Schutz vor Überlastung	Dienst stabil halten
Schutz vor Brute Force	Loginversuche begrenzen
Fairness	Ressourcen gleichmäßiger verteilen

WAF

WAF steht für Web Application Firewall.

Eine WAF schützt Webanwendungen vor bestimmten Angriffen.

Beispiele:

Angriff	Erklärung
SQL Injection	Manipulation von Datenbankabfragen
Cross-Site Scripting	Einschleusen von Skripten
Path Traversal	Zugriff auf unerlaubte Dateien
Brute Force	Viele Loginversuche
Exploit-Versuche	Ausnutzen bekannter Schwachstellen

Eine WAF ersetzt keine sichere Anwendung, kann aber zusätzlichen Schutz bieten.

Load Balancing in der DMZ

Öffentlich erreichbare Dienste stehen oft in einer DMZ.

Beispiel:

Zone	Systeme
Internet	Benutzer
DMZ	Load Balancer, Reverse Proxy
Internes Netz	Anwendung, Datenbank

Vorteil:

Interne Server sind nicht direkt aus dem Internet erreichbar.

Typische Webserver-Architektur

Beispiel:

Ebene	Systeme
Internet	Benutzer
DMZ	Reverse Proxy / Load Balancer
Webebene	WEB01, WEB02
Anwendungsebene	APP01, APP02
Datenbankebene	DB01
Backup	BKP01

Diese Trennung verbessert Sicherheit, Wartbarkeit und Skalierbarkeit.

Drei-Schichten-Architektur

Eine typische Anwendung kann aus drei Schichten bestehen.

Schicht	Aufgabe
Präsentationsschicht	Weboberfläche
Anwendungsschicht	Logik und Verarbeitung
Datenschicht	Datenbank und Speicher

Beispiel:

Schicht	Beispiel
Präsentation	WEB01, WEB02
Anwendung	APP01, APP02
Daten	DB01, DB02

Warum Datenbanken oft nicht einfach load-balanced werden

Webserver können häufig einfach vervielfacht werden.

Datenbanken sind schwieriger.

Grund:

Datenbanken müssen konsistente Daten liefern.

Probleme:

Problem	Erklärung
Schreibzugriffe	Änderungen dürfen nicht widersprüchlich sein
Transaktionen	Vorgänge müssen vollständig sein
Locks	Gleichzeitige Zugriffe müssen geregelt werden
Replikationsverzögerung	Kopien können kurzzeitig alt sein
Konflikte	Mehrere Schreibstellen sind schwierig

Caching

Caching bedeutet Zwischenspeichern.

Ziel:

Häufig benötigte Daten schneller bereitstellen.

Beispiele:

Cache	Beispiel
Browser-Cache	Bilder und CSS lokal speichern
Reverse-Proxy-Cache	Webseiten zwischenspeichern
Datenbank-Cache	Häufige Abfragen im RAM
Application-Cache	Ergebnisse zwischenspeichern
CDN	Inhalte weltweit verteilen

Caching kann Last reduzieren, ersetzt aber keine saubere Skalierung.

CDN

CDN steht für Content Delivery Network.

Ein CDN verteilt Inhalte über viele Standorte.

Typische Inhalte:

Inhalt	Beispiel
Bilder	Produktbilder
Videos	Streaming-Inhalte

Inhalt	Beispiel
JavaScript	Webdateien
CSS	Stylesheets
Downloads	Dateien

Vorteile:

Vorteil	Erklärung
Schnellere Auslieferung	Nähe zum Benutzer
Weniger Last auf Ursprungsserver	CDN liefert Inhalte aus
Schutz bei Lastspitzen	Mehr Verteilung
Teilweise DDoS-Schutz	Je nach Anbieter

DDoS und Last

DDoS steht für Distributed Denial of Service.

Dabei wird ein Dienst mit sehr vielen Anfragen überlastet.

Load Balancing kann helfen, ist aber allein kein vollständiger DDoS-Schutz.

Maßnahmen:

Maßnahme	Erklärung
Rate Limiting	Anfragen begrenzen
CDN	Last verteilen
WAF	Webangriffe filtern
DDoS-Schutzdienst	Angriffstraffic abwehren
Monitoring	Angriff erkennen
Netzwerkfilter	Unerwünschten Traffic blockieren

Typische Fehler beim Load Balancing

Fehler	Auswirkung
Kein Health Check	Anfragen gehen an defekte Server
Load Balancer nicht redundant	Load Balancer wird Single Point of Failure
Sticky Sessions falsch geplant	Ungleichmäßige Last

Fehler	Auswirkung
Backends direkt öffentlich erreichbar	Sicherheitsrisiko
Keine gemeinsamen Sessions	Benutzer verlieren Sitzungen
TLS falsch konfiguriert	Sicherheitsproblem
Keine Logs	Fehler schwer nachvollziehbar
Keine Ressourcenreserve	Ausfall eines Backends überlastet andere
Datenbank als Engpass vergessen	Webserver skalieren, aber DB bleibt langsam
Kein Monitoring	Überlastung wird zu spät erkannt

Gute Praxis

Empfehlung	Grund
Health Checks einrichten	Defekte Backends erkennen
Load Balancer redundant betreiben	Kein Single Point of Failure
Backends nicht direkt öffentlich machen	Sicherheit erhöhen
TLS sauber konfigurieren	Sichere Kommunikation
Sessions zentral speichern	Besser skalierbar
Stateless-Anwendungen bevorzugen	Einfachere Lastverteilung
Monitoring einrichten	Last und Fehler erkennen
Ressourcenreserven einplanen	Ausfälle abfangen
Rolling Updates nutzen	Wartung ohne Komplettausfall
Datenbanklast beachten	Engpässe vermeiden
Dokumentation pflegen	Betrieb nachvollziehbar machen

Prüfungsnahes Beispiel 1

Aufgabe:

Eine Webseite läuft auf einem einzelnen Webserver. Bei vielen Zugriffen wird die Webseite langsam.

Frage:

Welche Maßnahme kann helfen?

Mögliche Antwort:

Es können mehrere Webserver betrieben und über einen Load Balancer angesprochen werden. Der Load Balancer verteilt die Anfragen auf die Webserver. Dadurch wird die Last verteilt und die Verfügbarkeit verbessert.

Prüfungsnahes Beispiel 2

Aufgabe:

Ein Unternehmen betreibt drei Webserver hinter einem Load Balancer. Einer der Webserver fällt aus.

Frage:

Was sollte der Load Balancer tun?

Mögliche Antwort:

Der Load Balancer sollte den Ausfall über Health Checks erkennen und keine neuen Anfragen mehr an den defekten Webserver senden. Die Anfragen werden auf die verbleibenden funktionierenden Webserver verteilt.

Prüfungsnahes Beispiel 3

Aufgabe:

Ein Webshop speichert Warenkörbe lokal auf dem Webserver. Benutzer verlieren manchmal ihren Warenkorb, wenn mehrere Webserver genutzt werden.

Frage:

Was ist vermutlich das Problem?

Mögliche Antwort:

Die Anwendung ist stateful und speichert Sitzungsdaten lokal. Wenn der Benutzer bei einer späteren Anfrage auf einem anderen Webserver landet, kennt dieser den Warenkorb nicht. Lösungen sind Sticky Sessions oder ein zentraler Session-Speicher.

Prüfungsnahes Beispiel 4

Aufgabe:

Ein Administrator sagt: „Wir skalieren unsere Datenbank einfach wie unsere Webserver, indem wir mehrere Datenbankserver parallel hinter einen Load Balancer hängen.“

Frage:

Warum ist das problematisch?

Mögliche Antwort:

Datenbanken müssen Daten konsistent halten. Schreibzugriffe, Transaktionen und gleichzeitige Änderungen sind komplex. Datenbanken benötigen spezielle Replikations-, Cluster- oder Failover-Konzepte und können nicht immer wie einfache Webserver horizontal skaliert werden.

Prüfungsnahes Beispiel 5

Aufgabe:

Ein Unternehmen hat zwei Load Balancer, aber nur einer besitzt die aktive virtuelle IP-Adresse. Fällt dieser aus, übernimmt der zweite.

Frage:

Welches Prinzip wird genutzt?

Mögliche Antwort:

Es wird ein redundanter Load-Balancer-Aufbau mit virtueller IP-Adresse genutzt. Häufig wird dafür ein Verfahren wie VRRP eingesetzt. Der zweite Load Balancer übernimmt bei Ausfall die VIP.

Typische Prüfungsfrage: Was ist Load Balancing?

Mögliche Antwort:

Load Balancing bedeutet Lastverteilung. Ein Load Balancer verteilt eingehende Anfragen auf mehrere Backend-Server, damit die Last besser verteilt und die Verfügbarkeit erhöht wird.

Typische Prüfungsfrage: Warum sind Health Checks wichtig?

Mögliche Antwort:

Health Checks prüfen, ob Backend-Server erreichbar und funktionsfähig sind. Wenn ein Server ausfällt, kann der Load Balancer ihn aus der Verteilung nehmen und Anfragen nur noch an funktionierende Server senden.

Typische Prüfungsfrage: Unterschied Failover und Load Balancing

Failover	Load Balancing

Übernahme bei Ausfall	Verteilung von Anfragen
Häufig eine aktive Ressource	Mehrere aktive Server
Ziel: Ausfallsicherheit	Ziel: Lastverteilung und Verfügbarkeit
Beispiel: VM startet auf anderem Host	Beispiel: Webanfragen auf mehrere Webserver

---

Typische Prüfungsfrage: Was ist Scale-Up?

Mögliche Antwort:

Scale-Up bedeutet vertikale Skalierung. Dabei wird ein einzelnes System leistungsfähiger gemacht, zum Beispiel durch mehr CPU, mehr RAM, schnelleren Storage oder eine größere VM.

Typische Prüfungsfrage: Was ist Scale-Out?

Mögliche Antwort:

Scale-Out bedeutet horizontale Skalierung. Dabei werden weitere Systeme hinzugefügt, zum Beispiel mehrere Webserver hinter einem Load Balancer.

Typische Prüfungsfrage: Unterschied Scale-Up und Scale-Out

Scale-Up	Scale-Out
Einzelnes System stärker machen	Mehr Systeme hinzufügen
Mehr RAM, CPU, Storage	Weitere Server oder Instanzen
Einfacher, aber begrenzt	Skalierbarer, aber komplexer
Single Point of Failure bleibt möglich	Kann Verfügbarkeit verbessern

---

Typische Prüfungsfrage: Was bedeutet Sticky Session?

Mögliche Antwort:

Sticky Session bedeutet, dass ein Benutzer während einer Sitzung immer wieder demselben Backend-Server zugeordnet wird. Das ist nötig, wenn eine Anwendung Sitzungsdaten lokal auf dem Server speichert.

Typische Prüfungsfrage: Was bedeutet stateless?

Mögliche Antwort:

Stateless bedeutet, dass ein Server keinen wichtigen Sitzungszustand lokal speichert. Dadurch können Anfragen einfacher auf verschiedene Server verteilt werden, was Load Balancing und Skalierung erleichtert.

Typische Prüfungsfrage: Warum ist ein einzelner Load Balancer problematisch?

Mögliche Antwort:

Ein einzelner Load Balancer kann selbst ein Single Point of Failure sein. Wenn er ausfällt, ist der Dienst trotz mehrerer Backend-Server nicht erreichbar. Deshalb sollten Load Balancer redundant aufgebaut werden.

Typische Prüfungsfrage: Was ist ein Reverse Proxy?

Mögliche Antwort:

Ein Reverse Proxy nimmt Anfragen von Clients entgegen und leitet sie an interne Server weiter. Er kann zusätzlich TLS-Terminierung, Logging, Zugriffskontrolle und Load Balancing übernehmen.

Typische Prüfungsfrage: Was ist TLS-Terminierung?

Mögliche Antwort:

TLS-Terminierung bedeutet, dass der Load Balancer oder Reverse Proxy die HTTPS-Verbindung des Clients entgegennimmt und entschlüsselt. Danach leitet er die Anfrage intern an Backend-Server weiter.

Typische Prüfungsfrage: Warum sind Datenbanken schwieriger zu skalieren als Webserver?

Mögliche Antwort:

Datenbanken müssen Daten konsistent halten. Schreibzugriffe, Transaktionen und gleichzeitige Änderungen dürfen keine Widersprüche erzeugen. Deshalb benötigen Datenbanken spezielle Replikations- oder Clusterkonzepte.

Wichtige Begriffe

Begriff	Kurz erklärt
Load Balancing	Lastverteilung auf mehrere Server
Load Balancer	System, das Anfragen verteilt
Backend	Zielserver hinter dem Load Balancer

Begriff	Kurz erklärt
Frontend	Adresse oder Dienst, den Clients ansprechen
VIP	Virtuelle IP-Adresse
Health Check	Prüfung eines Backends
Round Robin	Verteilung der Reihe nach
Least Connections	Server mit wenigsten Verbindungen wird bevorzugt
Weighted Round Robin	Gewichtete Verteilung
IP Hash	Zielserver anhand Client-IP
Sticky Session	Benutzer bleibt auf gleichem Backend
Session Affinity	Sitzung wird Backend zugeordnet
Stateless	Zustandslos
Stateful	Zustandsbehaftet
Scale-Up	Einzelnes System stärker machen
Scale-Out	Weitere Systeme hinzufügen
Reverse Proxy	Vermittler zwischen Client und internen Servern
TLS-Terminierung	HTTPS wird am Proxy/Load Balancer beendet
VRRP	Virtuelle IP für Redundanz
Keepalived	Linux-Software für VRRP
Rolling Update	Systeme nacheinander aktualisieren
Blue-Green-Deployment	Umschaltung zwischen zwei Umgebungen
Canary Deployment	Neue Version zuerst für wenige Benutzer
Autoscaling	Automatische Skalierung
Bottleneck	Engpass
Throughput	Durchsatz
Antwortzeit	Zeit bis zur Antwort
WAF	Web Application Firewall
CDN	Content Delivery Network

Wichtige Merksätze

Load Balancing verteilt Anfragen auf mehrere Server.

Load Balancing verbessert Lastverteilung und kann die Verfügbarkeit erhöhen.

Ein Load Balancer braucht Health Checks, damit defekte Backends erkannt werden.

Ein einzelner Load Balancer kann selbst ein Single Point of Failure sein.

Scale-Up bedeutet: ein System stärker machen.

Scale-Out bedeutet: mehrere Systeme hinzufügen.

Webserver lassen sich oft leichter horizontal skalieren als Datenbanken.

Stateless-Anwendungen sind einfacher zu skalieren als stateful Anwendungen.

Sticky Sessions können nötig sein, verschlechtern aber manchmal die Lastverteilung.

Ein Reverse Proxy kann auch als Load Balancer arbeiten.

Monitoring ist notwendig, um Engpässe und Überlastung zu erkennen.

Kurzzusammenfassung

Load Balancing und Skalierung helfen dabei, Dienste leistungsfähiger und besser verfügbar zu machen.

Ein Load Balancer verteilt Anfragen auf mehrere Backend-Server.

Wichtige Ziele sind:

Ziel	Bedeutung
Lastverteilung	Arbeit auf mehrere Server verteilen
Verfügbarkeit	Defekte Server aus der Verteilung nehmen
Skalierbarkeit	Weitere Server hinzufügen
Wartbarkeit	Server einzeln warten
Performance	Antwortzeiten verbessern

Skalierung gibt es in zwei Grundformen:

Art	Bedeutung
Scale-Up	Einzelnes System stärker machen
Scale-Out	Mehr Systeme hinzufügen

Für die IHK ist besonders wichtig:

Prüfungsrelevanter Punkt	Bedeutung

Load Balancing erklären	Anfragen werden verteilt
Health Checks verstehen	Defekte Backends erkennen
Failover vs. Load Balancing unterscheiden	Übernahme vs. Verteilung
Scale-Up vs. Scale-Out unterscheiden	Stärker machen vs. mehr Systeme
Stateful vs. Stateless verstehen	Sitzungszustand wichtig für Skalierung
Sticky Sessions kennen	Benutzer bleibt auf gleichem Server
Load Balancer als SPOF erkennen	Redundanz nötig
Datenbank-Skalierung einordnen	Komplexer als Webserver

Der nächste logische Schritt ist:

Prüfungszusammenfassung – Virtualisierung und Cluster
