13. Linux im Serverbetrieb

13.1 Linux im Serverbetrieb

Linux wird sehr häufig im Serverbetrieb eingesetzt.

Typische Einsatzbereiche sind:

- Webserver
- Datenbankserver
- Dateiserver
- DNS-Server
- DHCP-Server
- Mailserver
- Container-Hosts
- Monitoring-Systeme
- Backup-Server
- Proxy-Server
- VPN-Server
- NAS-Systeme

Für Fachinformatiker Systemintegration ist Linux im Serverbetrieb besonders wichtig, weil viele Dienste in Unternehmen, Rechenzentren, Cloud-Umgebungen und Homelabs auf Linux laufen.

Merksatz:

> Linux ist eines der wichtigsten Betriebssysteme im Serverbereich.

Lernziele

Nach dieser Seite solltest du erklären können:

- warum Linux häufig als Serverbetriebssystem genutzt wird
- welche Aufgaben Linux-Server übernehmen können
- was Serverdienste sind
- wie Serverdienste verwaltet werden
- welche Rolle SSH im Serverbetrieb hat
- warum Headless-Betrieb typisch ist
- wie Linux mit Webservern,
  Datenbanken,

Containern
und Freigaben zusammenhängt
- welche Grundregeln für sicheren Serverbetrieb gelten
- wie man typische Serverprobleme eingrenzt

Grundidee

Ein Server ist ein System, das Dienste für andere Systeme bereitstellt.

Ein Linux-Server kann zum Beispiel:

Webseiten ausliefern

Datenbanken bereitstellen

Dateien speichern

Benutzer authentifizieren

Container ausführen

Backups speichern

DNS-Namen auflösen

Netzwerkzugriffe absichern

Anwendungen hosten

Wichtig:

Ein Server ist nicht automatisch ein bestimmter Gerätetyp.
Server bedeutet vor allem:
Das System stellt Dienste bereit.

Merksatz:

Server stellen Dienste für Clients bereit.

Warum Linux im Serverbetrieb häufig genutzt wird

Linux wird auf Servern häufig eingesetzt, weil es:

- stabil läuft
- gut automatisierbar ist
- ressourcenschonend betrieben werden kann
- ohne grafische Oberfläche auskommt
- viele Netzwerkdienste unterstützt
- gut per SSH administrierbar ist
- stark in Cloud- und Containerumgebungen verbreitet ist
- viele Paketmanager und Repositories bietet
- gut dokumentierbare Konfigurationen nutzt
- flexibel anpassbar ist

Merksatz:

Linux ist stabil,
flexibel
und gut automatisierbar.

Server und Client

Ein Client nutzt einen Dienst.

Ein Server stellt einen Dienst bereit.

Beispiele:

Client	Serverdienst
Browser	Webserver
E-Mail-Programm	Mailserver
Datenbankanwendung	Datenbankserver
PC im Netzwerk	Dateiserver
SSH-Client	SSH-Server
DNS-Client	DNS-Server

Merksatz:

Client fragt an.

Server antwortet mit einem Dienst.

Serverdienst

Ein Serverdienst ist ein Programm, das im Hintergrund läuft und eine bestimmte Funktion bereitstellt.

Beispiele:

Dienst	Zweck
sshd	Fernadministration
nginx	Webserver
apache2	Webserver
mariadb	Datenbank
postgresql	Datenbank
named / bind9	DNS
isc-dhcp-server	DHCP
samba	SMB-Dateifreigaben
nfs-server	NFS-Dateifreigaben
docker	Container
cron	zeitgesteuerte Aufgaben

Merksatz:

Ein Server besteht oft aus mehreren Diensten.

Headless-Betrieb

Viele Linux-Server laufen ohne Monitor, Tastatur und grafische Oberfläche.

Das nennt man:

headless

Administration erfolgt meist über:

```
SSH

Weboberflächen

Automatisierung

Monitoring

Konfigurationsdateien
```

Vorteile:

```
weniger Ressourcenverbrauch

weniger Angriffsfläche

bessere Automatisierung

einfacher Remote-Betrieb
```

Merksatz:

```
Server werden häufig ohne grafische Oberfläche betrieben.
```

SSH im Serverbetrieb

SSH ist der wichtigste Zugriff für Linux-Serveradministration.

Typischer Port:

```
TCP 22
```

Beispiel:

```
ssh benutzer@server
```

SSH ermöglicht:

Befehle ausführen

Konfigurationen bearbeiten

Logs prüfen

Dienste verwalten

Dateien übertragen

Tunnel aufbauen

Wichtig:

SSH muss sicher konfiguriert werden.

Merksatz:

SSH ist der Standardzugang zur Linux-Serveradministration.

Typische Serveradministration über SSH

Beispiele:

```
systemctl status nginx

journalctl -u nginx

df -h

free -h

ss -tulpen

ip addr

ip route
```

```
nano /etc/nginx/nginx.conf

apt update

apt upgrade
```

Merksatz:

```
Linux-Server werden oft komplett über die Shell verwaltet.
```

Dienste verwalten

Auf vielen Linux-Systemen werden Dienste mit systemd verwaltet.

Wichtige Befehle:

Befehl	Bedeutung
systemctl status dienst	Status prüfen
systemctl start dienst	Dienst starten
systemctl stop dienst	Dienst stoppen
systemctl restart dienst	Dienst neu starten
systemctl reload dienst	Konfiguration neu laden
systemctl enable dienst	Autostart aktivieren
systemctl disable dienst	Autostart deaktivieren
journalctl -u dienst	Logs eines Dienstes anzeigen

Merksatz:

```
systemctl verwaltet Dienste.
journalctl zeigt Dienstlogs.
```

Autostart von Diensten

Ein Serverdienst soll oft automatisch beim Systemstart starten.

Autostart aktivieren:

```
systemctl enable dienst
```

Dienst sofort starten:

```
systemctl start dienst
```

Kombiniert:

```
systemctl enable --now dienst
```

Prüfen:

```
systemctl is-enabled dienst

systemctl is-active dienst
```

Merksatz:

```
enabled heißt:
startet beim Boot.
active heißt:
läuft gerade.
```

Webserver

Ein Webserver liefert Webseiten, Webanwendungen oder API-Antworten aus.

Typische Linux-Webserver:

```
nginx

Apache HTTP Server
```

Typische Ports:

Protokoll	Port
HTTP	TCP 80

Protokoll	Port
HTTPS	TCP 443

Typische Aufgaben:

```
statische Webseiten ausliefern

Reverse Proxy

TLS beenden

Weiterleitungen

Zugriffskontrolle

Logging
```

Merksatz:

```
Webserver stellen Inhalte über HTTP oder HTTPS bereit.
```

Nginx

Nginx wird häufig genutzt als:

```
Webserver

Reverse Proxy

TLS-Endpunkt

Load Balancer

Proxy vor Anwendungen
```

Typische Prüfungen:

```
systemctl status nginx

nginx -t

journalctl -u nginx

ss -tulpen

curl -I http://localhost
```

Merksatz:

```
nginx -t prüft die Konfiguration.
```

Apache

Apache HTTP Server wird häufig genutzt als:

```
klassischer Webserver

PHP-Webserver

virtueller Host-Server

Webserver für viele Anwendungen
```

Typische Prüfungen:

```
systemctl status apache2

apachectl configtest

journalctl -u apache2

ss -tulpen
```

Merksatz:

```
Apache und Nginx sind typische Webserver unter Linux.
```

Reverse Proxy

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

Beispiel:

```
Client → Reverse Proxy → interne Webanwendung
```

Typische Aufgaben:

```
TLS-Zertifikate verwalten

mehrere Anwendungen über eine IP erreichbar machen

Weiterleitung nach Hostname

Schutz interner Dienste

Logging

zentrale Zugriffskontrolle
```

Merksatz:

```
Reverse Proxy steht vor internen Diensten.
```

Datenbankserver

Linux wird häufig für Datenbanken genutzt.

Beispiele:

Datenbank	Typ
PostgreSQL	relationale Datenbank
MariaDB	relationale Datenbank
MySQL	relationale Datenbank

Datenbank	Typ
Redis	In-Memory-Datenbank / Cache
SQLite	dateibasierte Datenbank

Typische Aufgaben:

Daten speichern

Anwendungen versorgen

Benutzerrechte verwalten

Backups erstellen

Performance überwachen

Merksatz:

Datenbanken sind zentrale Dienste und brauchen besondere Sicherung.

Datenbankserver sicher betreiben

Wichtige Punkte:

nicht unnötig öffentlich erreichbar machen

starke Passwörter verwenden

Benutzerrechte begrenzen

regelmäßige Backups erstellen

Restore testen

Updates einspielen

Logs prüfen

```
Speicherplatz überwachen

Dienststatus überwachen
```

Merksatz:

```
Datenbankserver brauchen Sicherheit,
Backups
und Monitoring.
```

Dateiserver

Linux kann Dateien im Netzwerk bereitstellen.

Typische Protokolle:

Protokoll	Typischer Einsatz
SMB/CIFS	Windows- und gemischte Netzwerke
NFS	Linux/Unix-Umgebungen
SFTP	Dateiübertragung über SSH
rsync	Synchronisation und Backups

Typische Dienste:

```
samba

nfs-server

sshd
```

Merksatz:

```
Dateiserver stellen Speicher im Netzwerk bereit.
```

Samba

Samba ermöglicht SMB/CIFS-Freigaben unter Linux.

Typische Nutzung:

```
Windows-Freigaben

NAS-Freigaben

gemischte Netzwerke

gemeinsame Ordner
```

Wichtige Themen:

```
Benutzerrechte

Dateirechte

Freigaberechte

Authentifizierung

Netzwerkerreichbarkeit
```

Merksatz:

```
Samba verbindet Linux mit SMB/Windows-Freigaben.
```

NFS

NFS steht für:

```
Network File System
```

Typische Nutzung:

```
Linux-zu-Linux-Freigaben

Server-zu-Server-Speicher
```

```
virtuelle Umgebungen

Backup-Ziele
```

Wichtig:

```
UID/GID,
Export-Regeln
und Dateirechte müssen passen.
```

Merksatz:

```
NFS ist typisch für Linux/Unix-Dateifreigaben.
```

DNS-Server

Ein DNS-Server löst Namen in IP-Adressen auf.

Beispiel:

```
server01.intern
```

wird zu:

```
192.168.10.20
```

Typische Aufgaben:

```
interne Namensauflösung

Zonen verwalten

Weiterleitung an externe DNS-Server

Reverse DNS

DNS-Caching
```

Typische Software:

```
BIND

Unbound

dnsmasq
```

Merksatz:

```
DNS macht Namen zu IP-Adressen.
```

DHCP-Server

Ein DHCP-Server vergibt Netzwerkkonfigurationen automatisch.

Typische Informationen:

```
IP-Adresse

Subnetzmaske

Gateway

DNS-Server

Lease-Zeit
```

Vorteil:

```
Clients müssen nicht manuell konfiguriert werden.
```

Merksatz:

```
DHCP verteilt IP-Konfigurationen automatisch.
```

Container-Host

Linux ist sehr häufig die Grundlage für Container.

Typische Werkzeuge:

```
Docker

Podman

containerd
```

Container nutzen Linux-Funktionen wie:

```
Namespaces

cgroups

Dateisystem-Layer

Netzwerk-Bridges

Volumes
```

Typische Aufgaben:

```
Container starten

Images aktualisieren

Volumes sichern

Logs prüfen

Ports verwalten

Netzwerke konfigurieren
```

Merksatz:

```
Container laufen häufig auf Linux-Hosts.
```

Container und Ports

Containerdienste sind oft über Portweiterleitungen erreichbar.

Beispielidee:

```
Host-Port 8080 → Container-Port 80
```

Wichtig prüfen:

```
welcher Host-Port genutzt wird

welcher Container-Port genutzt wird

ob der Dienst im Container läuft

ob der Container im richtigen Netzwerk ist

ob Volumes korrekt eingebunden sind

ob Firewall-Regeln passen
```

Merksatz:

```
Bei Containern immer Host-Port,
Container-Port,
Netzwerk
und Volume unterscheiden.
```

Container und persistente Daten

Container selbst sind oft ersetzbar.

Daten sollten deshalb in:

```
Volumes

Bind Mounts
```

Datenbanken

externem Speicher

liegen.

Warum?

Container können neu erstellt werden

Images können aktualisiert werden

Container-Dateisysteme können verloren gehen

Backups müssen Daten außerhalb des Containers erfassen

Merksatz:

Container ersetzen,
Daten erhalten.

Serverrollen

Ein Linux-Server kann eine oder mehrere Rollen haben.

Beispiele:

Rolle	Aufgabe
Webserver	Webseiten bereitstellen
Datenbankserver	Daten speichern
Dateiserver	Dateien bereitstellen
Backupserver	Sicherungen speichern
Monitoringserver	Systeme überwachen
Proxyserver	Anfragen weiterleiten
Containerhost	Container ausführen
VPN-Server	sichere Tunnel bereitstellen
DNS-Server	Namen auflösen

Rolle	Aufgabe
DHCP-Server	IP-Konfiguration verteilen

Merksatz:

```
Serverrollen beschreiben,
welche Aufgabe ein Server übernimmt.
```

Mehrere Dienste auf einem Server

Ein Server kann mehrere Dienste ausführen.

Beispiel:

```
Webserver

Datenbank

Backup-Agent

Monitoring-Agent

SSH
```

Vorteile:

```
weniger Systeme nötig

einfache kleine Umgebung
```

Nachteile:

```
höhere Abhängigkeit

mehr Angriffsfläche

schwierigeres Troubleshooting
```

Ressourcen teilen sich mehrere Dienste

ein Ausfall betrifft mehrere Funktionen

Merksatz:

Mehr Dienste auf einem Server bedeuten mehr Abhängigkeiten.

Trennung von Diensten

In größeren Umgebungen werden Dienste oft getrennt.

Beispiel:

Webserver auf Server A

Datenbank auf Server B

Backup auf Server C

Monitoring auf Server D

Vorteile:

bessere Sicherheit

klarere Zuständigkeiten

einfachere Skalierung

bessere Fehlereingrenzung

weniger gegenseitige Beeinflussung

Nachteile:

mehr Systeme

mehr Netzwerkabhängigkeiten

mehr Administration

Merksatz:

Diensttrennung verbessert Sicherheit und Skalierbarkeit,
erhöht aber Verwaltungsaufwand.

Produktivsystem, Testsystem und Entwicklungssystem

Typische Umgebungen:

Umgebung	Zweck
Entwicklung	neue Funktionen entwickeln
Test	Änderungen prüfen
Produktion	echter Betrieb für Benutzer
Staging	produktionsnahe Vorstufe

Wichtig:

Änderungen sollten nicht ungetestet direkt in Produktion erfolgen.

Merksatz:

Produktion ist nicht der Ort für ungetestete Experimente.

Serverhärtung

Serverhärtung bedeutet, ein System sicherer zu konfigurieren und unnötige Risiken zu reduzieren.

Maßnahmen:

nur benötigte Dienste installieren

unnötige Ports schließen

```
Updates einspielen

SSH absichern

Firewall aktivieren

Benutzerrechte begrenzen

Logs überwachen

Backups einrichten

Standardpasswörter entfernen

Konfiguration dokumentieren
```

Merksatz:

```
Härtung reduziert Angriffsfläche.
```

Minimale Installation

Ein Server sollte nur Software enthalten, die für seine Aufgabe notwendig ist.

Vorteile:

```
weniger Angriffsfläche

weniger Updates

weniger Abhängigkeiten

weniger Fehlerquellen

bessere Übersicht
```

Beispiel:

Ein Datenbankserver braucht normalerweise keine grafische Desktopumgebung.

Merksatz:

Installiere nur,
was benötigt wird.

Firewall im Serverbetrieb

Ein Server sollte nur notwendige Ports erreichbar machen.

Beispiele:

Dienst	Port
SSH	TCP 22
HTTP	TCP 80
HTTPS	TCP 443
DNS	TCP/UDP 53
DHCP	UDP 67/68
SMB	TCP 445
PostgreSQL	TCP 5432
MySQL/MariaDB	TCP 3306

Wichtig:

Nicht jeder Dienst muss aus jedem Netz erreichbar sein.

Merksatz:

Ports nur für notwendige Quellen öffnen.

Monitoring im Serverbetrieb

Server sollten überwacht werden.

Wichtige Messwerte:

```
CPU

RAM

Swap

Speicherplatz

Inodes

Netzwerk

Dienststatus

offene Ports

Logs

Zertifikate

Backups

Antwortzeiten
```

Merksatz:

```
Monitoring erkennt Probleme,
bevor sie kritisch werden.
```

Backup im Serverbetrieb

Server brauchen regelmäßige Backups.

Zu sichern sind oft:

```
Anwendungsdaten

Datenbanken
```

Konfigurationen

Skripte

Zertifikate

Schlüssel

systemd-Units

Docker Compose Dateien

Dokumentation

Wichtig:

Restore testen

RPO und RTO definieren

Backup-Logs prüfen

externe Kopie vorhalten

Merksatz:

Backup ohne Restore-Test ist unsicher.

Dokumentation im Serverbetrieb

Dokumentation sollte enthalten:

Servername

IP-Adresse

Aufgabe

installierte Dienste

offene Ports

Benutzer und Berechtigungen

wichtige Pfade

Backup-Plan

Restore-Anleitung

Abhängigkeiten

Monitoring

Ansprechpartner

Änderungsverlauf

Merksatz:

> Gute Dokumentation spart im Fehlerfall Zeit.

Typische wichtige Pfade auf Servern

Pfad	Bedeutung
/etc	Konfigurationen
/var/log	Logs
/var/lib	Anwendungsdaten
/srv	bereitgestellte Dienstdaten
/opt	optionale Anwendungen
/home	Benutzerverzeichnisse
/root	Root-Home
/tmp	temporäre Dateien
/usr/bin	Programme

Pfad	Bedeutung
/etc/systemd/system	eigene systemd-Units

Merksatz:

```
Serverwissen bedeutet auch:
wichtige Pfade kennen.
```

Typische Serverprüfung nach Neustart

Nach einem Neustart prüfen:

```
Erreichbarkeit

SSH-Zugriff

wichtige Dienste

offene Ports

Speicherplatz

Logs

Mounts

Datenbanken

Webanwendungen

Backups
```

Beispielbefehle:

```
systemctl status dienst

ss -tulpen
```

```
df -h

journalctl -p err -b

findmnt
```

Merksatz:

```
Nach Neustart nicht nur schauen,
ob der Server pingt.
```

Typische Serverprüfung nach Update

Nach Updates prüfen:

```
Paketmanager meldet keine Fehler

Kernel-Neustart nötig?

Dienste laufen

Ports lauschen

Anwendungen antworten

Logs enthalten keine neuen Fehler

Speicherplatz ausreichend

Monitoring wieder grün
```

Beispielbefehle:

```
systemctl --failed

journalctl -p err -b

ss -tulpen
```

```
df -h
```

Merksatz:

```
Update fertig heißt erst fertig,
wenn Dienste und Anwendungen geprüft sind.
```

Typische Fehlerbilder im Serverbetrieb

Fehlerbild	Mögliche Ursache
Server nicht erreichbar	Netzwerk, Firewall, ausgeschaltet, Routing
SSH geht nicht	Dienst, Port, Firewall, Benutzer, Schlüssel
Webseite nicht erreichbar	Webserver, Port, DNS, TLS, Reverse Proxy
Datenbank nicht erreichbar	Dienst, Port, Bind-Adresse, Rechte
Dienst startet nicht	Konfiguration, Rechte, Abhängigkeit, Port
Speicher voll	Logs, Datenbanken, Backups, Container
hohe Last	CPU, RAM, I/O, Prozesse, Angriff
Mount fehlt	fstab, Netzwerk, Datenträger
Backup fehlt	Cron, Rechte, Ziel voll, Netzwerk
Name löst nicht auf	DNS, /etc/hosts, Resolver

Fehlersuche bei Serverdiensten

Sinnvolle Reihenfolge:

```
1. Fehlerbild genau beschreiben

2. Dienststatus prüfen

3. Logs prüfen

4. Port prüfen

5. lokale Verbindung testen
```

```
6. externe Verbindung testen

7. Firewall prüfen

8. Konfiguration prüfen

9. Rechte prüfen

10. Abhängigkeiten prüfen
```

Beispiel:

```
systemctl status nginx

journalctl -u nginx

ss -tulpen

curl -I http://127.0.0.1
```

Merksatz:

```
Dienstprobleme immer von innen nach außen prüfen.
```

Von innen nach außen prüfen

Bei einem Webdienst:

```
1. Läuft der Prozess?

2. Lauscht der Port lokal?

3. Antwortet der Dienst lokal?

4. Antwortet der Dienst über Server-IP?

5. Antwortet der Dienst aus dem Netzwerk?
```

6. Funktioniert DNS?

7. Funktioniert HTTPS?

8. Funktioniert die Anwendung?

Merksatz:

Erst lokal prüfen,
dann Netzwerk,
dann DNS,
dann Anwendung.

Typische Prüfungsfragen

Frage	Kurzantwort
Warum wird Linux häufig als Server genutzt?	stabil, flexibel, automatisierbar, ressourcenschonend
Was ist ein Serverdienst?	Hintergrunddienst, der eine Funktion bereitstellt
Was bedeutet headless?	Betrieb ohne Monitor und grafische Oberfläche
Wofür wird SSH genutzt?	sichere Fernadministration
Was ist ein Webserver?	Dienst für HTTP/HTTPS
Was ist ein Reverse Proxy?	vorgeschalteter Proxy vor internen Diensten
Was ist ein Datenbankserver?	Server zur strukturierten Datenspeicherung
Was ist ein Dateiserver?	stellt Dateien im Netzwerk bereit
Was ist ein Container-Host?	System, das Container ausführt
Warum Serverhärtung?	Angriffsfläche reduzieren
Warum Monitoring?	Probleme früh erkennen
Warum Dokumentation?	Betrieb und Fehlerbehebung nachvollziehbar machen

---

Typische Prüfungsfallen

Falle	Richtig denken
Server als Hardware definieren	Server ist vor allem eine Rolle
Dienst läuft = Dienst erreichbar	Port, Firewall, Bind-Adresse prüfen
Ping geht = Anwendung geht	Ping prüft nicht HTTP, SSH oder Datenbank

Falle	Richtig denken
Headless mit eingeschränkter Verwaltung verwechseln	Verwaltung erfolgt per SSH und Tools
Datenbank öffentlich erreichbar machen	nur notwendige Quellen erlauben
Containerdaten im Container speichern	Volumes oder Bind Mounts nutzen
Backup nur für Benutzerdaten planen	auch Konfigurationen sichern
Update ohne Nachprüfung	Dienste, Logs und Anwendungen prüfen
mehrere Dienste ohne Abhängigkeiten betrachten	Dienste beeinflussen sich gegenseitig
keine Dokumentation	Fehlerfall wird langsamer und riskanter

IHK-sichere Kurzformulierung

Linux wird häufig als Serverbetriebssystem eingesetzt, weil es stabil, ressourcenschonend, gut automatisierbar und per SSH administrierbar ist. Ein Server stellt Dienste für Clients bereit, zum Beispiel Webserver, Datenbankserver, Dateiserver, DNS, DHCP, Container oder Backupdienste. Viele Linux-Server laufen headless, also ohne grafische Oberfläche, und werden über SSH, systemctl, journalctl und Konfigurationsdateien verwaltet. Im sicheren Serverbetrieb sind minimale Installation, regelmäßige Updates, Firewall-Regeln, Diensthärtung, Monitoring, Backups, Restore-Tests und Dokumentation wichtig. Bei Fehlern wird systematisch geprüft: Dienststatus, Logs, offene Ports, lokale Antwort, Netzwerkzugriff, Firewall, Konfiguration, Rechte und Abhängigkeiten.

Merksätze

Linux ist im Serverbereich sehr verbreitet.

Server stellen Dienste bereit.

Client fragt an,
Server antwortet.

Serverdienst = Hintergrunddienst mit Aufgabe.

Headless bedeutet:
ohne grafische Oberfläche.

SSH ist Standard für Fernadministration.

systemctl verwaltet Dienste.

journalctl zeigt Dienstlogs.

enabled heißt Autostart.

active heißt läuft gerade.

Webserver nutzen HTTP und HTTPS.

HTTP nutzt TCP 80.

HTTPS nutzt TCP 443.

Reverse Proxy steht vor internen Diensten.

Datenbanken brauchen besondere Sicherung.

Dateiserver stellen Speicher im Netzwerk bereit.

Samba nutzt SMB/CIFS.

NFS ist typisch für Linux/Unix-Freigaben.

DNS löst Namen auf.

DHCP verteilt IP-Konfiguration.

Container laufen häufig auf Linux-Hosts.

Containerdaten gehören in Volumes oder Bind Mounts.

Serverrollen beschreiben Aufgaben.

Mehr Dienste bedeuten mehr Abhängigkeiten.

Diensttrennung verbessert Sicherheit und Skalierbarkeit.

Produktion ist nicht für ungetestete Experimente.

Härtung reduziert Angriffsfläche.

Nur benötigte Dienste betreiben.

Nur notwendige Ports öffnen.

Monitoring erkennt Probleme früh.

Backup ohne Restore-Test ist unsicher.

Dokumentation spart im Fehlerfall Zeit.

Dienstprobleme von innen nach außen prüfen.