4.5 Dienstabhängigkeiten untersuchen*

Ein Dienst kann korrekt gestartet sein und trotzdem nicht funktionieren, wenn eine benötigte Abhängigkeit fehlt oder fehlerhaft arbeitet. Solche Abhängigkeiten können vom Betriebssystem ausdrücklich konfiguriert oder nur innerhalb der Anwendung hinterlegt sein.

> **Grundsatz:**
> Ein abhängiger Dienst kann nur so zuverlässig funktionieren wie die gesamte Kette seiner benötigten Komponenten.

Ziele dieser Seite

Nach dieser Seite sollst du:

- formale und versteckte Dienstabhängigkeiten unterscheiden können,
- erforderliche und optionale Abhängigkeiten erkennen können,
- Startreihenfolge und echte Abhängigkeit auseinanderhalten können,
- vorwärts und rückwärts gerichtete Abhängigkeiten ermitteln können,
- DNS-, Netzwerk-, Speicher-, Datenbank- und Authentifizierungsabhängigkeiten prüfen können,
- Fehler entlang einer Dienstkette eingrenzen können,
- gemeinsame Abhängigkeiten bei mehreren gestörten Diensten erkennen können,
- Auswirkungen eines Dienstneustarts auf abhängige Systeme bewerten können.

---

1. Formale und funktionale Abhängigkeiten unterscheiden

Art	Beschreibung	Beispiel
Formale Dienstabhängigkeit	In der Betriebssystem-Dienstverwaltung eingetragen	Webdienst benötigt einen lokalen Datenbankdienst
Startreihenfolge	Legt fest, was vorher oder nachher gestartet wird	Anwendung startet nach dem Netzwerkziel
Netzwerkabhängigkeit	Externes System muss über das Netzwerk erreichbar sein	Anwendung verbindet sich mit Datenbankserver
Namensauflösung	Dienst benötigt funktionierendes DNS	Backend wird über Hostnamen angesprochen
Speicherabhängigkeit	Lokales oder entferntes Dateisystem muss verfügbar sein	Anwendung benötigt SMB- oder NFS-Freigabe

Art	Beschreibung	Beispiel
Authentifizierungsabhängigkeit	Identitätsdienst wird benötigt	Anmeldung über Active Directory oder LDAP
Zertifikatsabhängigkeit	Vertrauenskette und Gültigkeit müssen stimmen	TLS-Verbindung zu einer API
Zeitabhängigkeit	Systemuhren müssen ausreichend synchron sein	Kerberos oder Zertifikatsprüfung
Ressourcenabhängigkeit	CPU, RAM, Speicherplatz oder Dateideskriptoren werden benötigt	Datenbank kann keine Dateien mehr schreiben
Anwendungsabhängigkeit	Andere Anwendung oder API muss funktionieren	Warenwirtschaft greift auf Zahlungsdienst zu
Konfigurationsabhängigkeit	Datei, Variable oder Secret muss vorhanden sein	Datenbank-URL aus Umgebungsvariable
Infrastrukturabhängigkeit	Plattformkomponente stellt Laufzeit bereit	Container-Runtime, Hypervisor oder Cluster

> Betriebssystemwerkzeuge zeigen normalerweise nur die formal registrierten Abhängigkeiten. Externe Datenbanken, DNS-Server, APIs oder Netzwerkspeicher erscheinen dort häufig nicht.

---

2. Eine Dienstkette darstellen

Eine Anwendung kann beispielsweise folgende Kette besitzen:

```
Benutzer
  → Client
  → DNS
  → Netzwerk und Firewall
  → Loadbalancer
  → Reverse Proxy
  → Webserver
  → Anwendungsdienst
  → Datenbank
  → Verzeichnisdienst
  → Netzwerkspeicher
  → externe API
```

Für jede Verbindung sollten folgende Informationen erfasst werden:

Eigenschaft	Beispiel
Quellsystem	Anwendungsserver
Zielsystem	Datenbankserver
Zielname	`srv-db01.example.local`
Zielport	TCP 5432
Protokoll	PostgreSQL
Authentifizierung	Dienstkonto
Verschlüsselung	TLS
Zeitüberschreitung	10 Sekunden
Kritikalität	Erforderlich
Verhalten bei Ausfall	Anmeldung nicht möglich
Überwachung	TCP- und Datenbankabfrage
Verantwortlichkeit	Datenbankbetrieb

Praktische Dokumentationsform:

Quelle	Abhängigkeit	Ziel	Port/Protokoll	Kritisch	Nachweis
Webserver	Namensauflösung	DNS-Server	UDP/TCP 53	Ja	DNS-Abfrage
Webserver	Anwendung	App-Server	TCP 8443	Ja	HTTPS-Test
App-Server	Datenbank	DB-Server	TCP 5432	Ja	Datenbankabfrage
App-Server	Anmeldung	LDAP-Server	TCP 636	Ja	LDAPS-Test
App-Server	Dateispeicher	Fileserver	SMB 445	Nein	Freigabetest

---

3. Erforderliche und optionale Abhängigkeiten unterscheiden

Abhängigkeitstyp	Verhalten bei Ausfall
Zwingend erforderlich	Dienst kann nicht starten oder Hauptfunktion fällt aus
Optional	Nur Zusatzfunktion fällt aus
Bedingt erforderlich	Nur bestimmte Benutzer oder Vorgänge sind betroffen

Abhängigkeitstyp	Verhalten bei Ausfall
Redundant	Andere Instanz kann übernehmen
Zwischengespeichert	Funktioniert vorübergehend mit vorhandenen Cache-Daten
Asynchron	Anfragen werden zunächst in eine Warteschlange geschrieben
Startabhängig	Nur während des Starts erforderlich
Laufzeitabhängig	Während des gesamten Betriebs erforderlich
Verwaltungsabhängig	Nur für Administration oder Monitoring erforderlich

Beispiele:

- Eine Webanwendung kann ihre Startseite ohne Datenbank ausliefern, aber keine Benutzer anmelden.
- Ein Maildienst kann Nachrichten zwischenspeichern, obwohl das Zielsystem vorübergehend nicht erreichbar ist.
- Ein Cluster kann beim Ausfall einer Instanz weiterarbeiten.
- Ein Dienst kann starten, obwohl ein optionales Monitoring-Backend fehlt.
- Eine Anwendung kann nach dem Start weiterarbeiten, obwohl der Konfigurationsdienst später ausfällt.

> Der Ausfall einer Abhängigkeit führt nicht immer zum vollständigen Dienststillstand. Teilfunktionen und verzögerte Fehler müssen deshalb ausdrücklich geprüft werden.

---

4. Windows-Dienstabhängigkeiten mit PowerShell prüfen

`<DIENSTNAME>` muss durch den internen Dienstnamen ersetzt werden.

Dienst und erforderliche Systemdienste anzeigen:

```
[RO] Get-Service -Name "<DIENSTNAME>" -RequiredServices
```

Nur die erforderlichen Dienste übersichtlich anzeigen:

```
[RO] (Get-Service -Name "<DIENSTNAME>").ServicesDependedOn |
    Select-Object Name, DisplayName, Status
```

Dienste anzeigen, die vom untersuchten Dienst abhängen:

```
[RO] Get-Service -Name "<DIENSTNAME>" -DependentServices
```

Abhängige Dienste übersichtlich anzeigen:

```
[RO] (Get-Service -Name "<DIENSTNAME>").DependentServices |
    Select-Object Name, DisplayName, Status
```

Dienst und beide Abhängigkeitsrichtungen erfassen:

```
[RO] $service = Get-Service -Name "<DIENSTNAME>"

$service | Select-Object Name, DisplayName, Status

"Benötigte Dienste:"
$service.ServicesDependedOn |
    Select-Object Name, DisplayName, Status

"Abhängige Dienste:"
$service.DependentServices |
    Select-Object Name, DisplayName, Status
```

Wichtig:

- `ServicesDependedOn` zeigt Dienste, die der untersuchte Dienst benötigt.
- `DependentServices` zeigt Dienste, die den untersuchten Dienst benötigen.
- Beide Richtungen sind vor einem Stopp oder Neustart relevant.
- Nicht jede funktionale Anwendungsabhängigkeit ist beim Service Control Manager registriert.

---

5. Windows-Dienstabhängigkeiten mit sc.exe prüfen

Konfiguration einschließlich eingetragener Abhängigkeiten anzeigen:

```
[RO][SENS] sc.exe qc "<DIENSTNAME>"
```

Relevant ist insbesondere der Abschnitt:

```
DEPENDENCIES
```

Direkt abhängige Dienste anzeigen:

```
[RO] sc.exe enumdepend "<DIENSTNAME>"
```

Erweiterte Dienstinformationen anzeigen:

```
[RO] sc.exe queryex "<DIENSTNAME>"
```

Interpretation:

Abfrage	Richtung
`sc.exe qc` → `DEPENDENCIES`	Welche Dienste benötigt der untersuchte Dienst?
`sc.exe enumdepend`	Welche Dienste hängen vom untersuchten Dienst ab?

> “ `sc.exe qc` kann Programmpfade, Dienstkonten und interne Konfigurationsinformationen anzeigen. Die Ausgabe sollte deshalb als potenziell sensibel behandelt werden.

---

6. Zustand aller formalen Windows-Abhängigkeiten prüfen

Benötigte Dienste samt Startart und Konto untersuchen:

```
[RO] $requiredServices = (Get-Service -Name "<DIENSTNAME>").ServicesDependedOn

foreach ($requiredService in $requiredServices) {
```

```
    Get-CimInstance Win32_Service `
        -Filter "Name='$($requiredService.Name)'" |
        Select-Object Name,
                      DisplayName,
                      State,
                      StartMode,
                      StartName,
                      ProcessId,
                      ExitCode
}
```

Mögliche Auffälligkeiten:

- erforderlicher Dienst ist beendet,
- erforderlicher Dienst ist deaktiviert,
- Dienst hängt in `StartPending`,
- Dienst läuft unter einem falschen Konto,
- Dienstprozess besitzt keine PID,
- Exitcode ist ungleich null,
- benötigter Dienst wird wiederholt neu gestartet,
- Abhängigkeitsdefinition enthält einen nicht mehr vorhandenen Dienst.

> Ein laufender erforderlicher Dienst kann trotzdem funktional gestört sein. Nach dem Status müssen Listener, Protokoll und tatsächliche Funktion geprüft werden.

---

7. Abhängigkeiten unter Linux mit systemd anzeigen

Direkte und indirekte Abhängigkeiten anzeigen:

```
[RO] systemctl list-dependencies "<DIENST>" --no-pager
```

Nur unmittelbare Abhängigkeiten anzeigen:

```
[RO] systemctl list-dependencies \
  --plain \
```

```
  --no-pager \
  "<DIENST>"
```

Rückwärts gerichtete Abhängigkeiten anzeigen:

```
[RO] systemctl list-dependencies \
  --reverse \
  --no-pager \
  "<DIENST>"
```

Abhängigkeiten und Reihenfolgenbeziehungen strukturiert anzeigen:

```
[RO] systemctl show "<DIENST>" \
  --property=Requires,Wants,Requisite,BindsTo,PartOf,After,Before,Conflicts
```

Wirksame Unit-Konfiguration anzeigen:

```
[RO][FILE][SENS] systemctl cat "<DIENST>"
```

Status abhängiger Units prüfen:

```
[RO] systemctl --failed --no-pager
```

> “ `systemctl list-dependencies` zeigt systemd-Units. Eine in der Anwendung konfigurierte Datenbank oder API wird nur dann sichtbar, wenn dafür ausdrücklich eine systemd-Beziehung definiert wurde.

---

8. systemd-Beziehungen richtig interpretieren

Direktive	Grundbedeutung
`Requires=`	Starke Anforderungsbeziehung zu einer anderen Unit
`Wants=`	Schwächere Anforderungsbeziehung

Direktive	Grundbedeutung
`Requisite=`	Andere Unit muss bereits aktiv sein; sie wird dadurch nicht automatisch gestartet
`BindsTo=`	Engere Bindung an den Aktivzustand einer anderen Unit
`PartOf=`	Bestimmte Stop- und Neustartaktionen werden weitergegeben
`After=`	Diese Unit wird nach der genannten Unit gestartet
`Before=`	Diese Unit wird vor der genannten Unit gestartet
`Conflicts=`	Units sollen nicht gleichzeitig aktiv sein
`OnFailure=`	Andere Unit wird bei einem Fehlschlag aktiviert

Besonders wichtig:

```
Requires=  ist nicht dasselbe wie After=
Wants=     ist nicht dasselbe wie After=
After=     legt Reihenfolge fest, aber keine zwingende Anforderung
Before=    legt Reihenfolge fest, aber keine zwingende Anforderung
```

Beispiel:

```
[Unit]
Requires=postgresql.service
After=network-online.target postgresql.service
```

Mögliche Interpretation:

- Der Dienst besitzt eine starke Anforderung an `postgresql.service`.
- Er soll nach `postgresql.service` gestartet werden.
- Er soll außerdem nach `network-online.target` gestartet werden.
- Das Erreichen von `network-online.target` beweist nicht, dass eine bestimmte entfernte Datenbank tatsächlich erreichbar ist.

> Für eine zuverlässige Bewertung müssen Anforderungs- und Reihenfolgedirektiven gemeinsam betrachtet werden.

9. Abhängigkeitsbaum unter systemd in beide Richtungen lesen

Vorwärts gerichtete Frage:

> Was benötigt dieser Dienst?

```
[RO] systemctl list-dependencies "<DIENST>" --no-pager
```

Rückwärts gerichtete Frage:

> Welche Units benötigen diesen Dienst?

```
[RO] systemctl list-dependencies \
  --reverse \
  --no-pager \
  "<DIENST>"
```

Nur Abhängigkeiten vor einem Neustart prüfen:

```
[RO] systemctl show "<DIENST>" \
  --property=RequiredBy,WantedBy,ConsistsOf,PartOf
```

Bewertung vor einer Maßnahme:

Feststellung	Bedeutung
Viele rückwärtige Abhängigkeiten	Neustart kann mehrere Dienste beeinflussen
`PartOf=` vorhanden	Neustart- oder Stoppaktionen können gekoppelt sein
`BindsTo=` vorhanden	Ausfall kann abhängige Unit ebenfalls deaktivieren
Gemeinsame Datenbankabhängigkeit	Datenbankfehler kann mehrere Anwendungen betreffen
Gemeinsames Netzwerkmount	Speicherfehler kann mehrere Dienste blockieren
Gemeinsames Target	Nicht jede aufgeführte Unit ist zwingend funktional abhängig

10. Abhängigkeiten unter macOS mit launchd untersuchen

launchd besitzt laut Apple kein allgemeines ausdrückliches Abhängigkeitsmodell wie systemd. Dienstinteraktionen sollen möglichst über bedarfsgesteuerte IPC-Mechanismen gelöst werden.

Deshalb müssen mehrere Informationsquellen kombiniert werden.

Systemweiten Dienst anzeigen:

```
[RO] launchctl print "system/<LABEL>"
```

Benutzerbezogenen Dienst anzeigen:

```
[RO] launchctl print "gui/$(id -u)/<LABEL>"
```

Konfigurationsdatei eines bekannten Drittanbieterdienstes anzeigen:

```
[RO][FILE][SENS] plutil -p "/Library/LaunchDaemons/<LABEL>.plist"
```

Benutzerbezogenen LaunchAgent anzeigen:

```
[RO][FILE][SENS] plutil -p "$HOME/Library/LaunchAgents/<LABEL>.plist"
```

In der plist können unter anderem folgende Auslöser relevant sein:

Schlüssel	Bedeutung
`MachServices`	Vom Job bereitgestellte Mach-Dienste
`Sockets`	Von launchd verwaltete Sockets
`KeepAlive`	Bedingungen für fortlaufende oder erneute Ausführung
`PathState`	Startbedingung anhand vorhandener Pfade
`OtherJobEnabled`	Bedingung anhand des Aktivierungszustands anderer Jobs
`WatchPaths`	Start bei Änderung beobachteter Pfade
`QueueDirectories`	Start bei nicht leerem Verzeichnis
`NetworkState`	Historische grobe Netzwerkbedingung; nicht als Erreichbarkeitsnachweis verwenden
`StartOnMount`	Start beim Einhängen eines Dateisystems

> Der Schlüssel `OtherJobEnabled` bildet keine harte, zuverlässige Dienstabhängigkeit wie ein allgemeines Abhängigkeitsmodell. Apple beschreibt solche Bedingungen als Hinweise und empfiehlt bedarfsgesteuerte IPC-Mechanismen.

---

11. Versteckte Anwendungsabhängigkeiten finden

Funktionale Abhängigkeiten können an folgenden Stellen dokumentiert oder konfiguriert sein:

- Herstellerdokumentation,
- Architekturdiagramm,
- Betriebshandbuch,
- Konfigurationsdatei,
- Umgebungsvariable,
- Dienststartparameter,
- Containerdefinition,
- Reverse-Proxy-Konfiguration,
- DNS-Eintrag,
- Zertifikatskonfiguration,
- Mount-Konfiguration,
- Secret- oder Credential-Verwaltung,
- Anwendungsprotokoll,
- Überwachungssystem,
- Quellcode oder Deploymentbeschreibung.

Typische Konfigurationsbegriffe:

```
host
hostname
server
endpoint
url
uri
database
db_host
connection_string
ldap
smtp
```

```
proxy
upstream
backend
api
mount
share
certificate
key
timeout
retry
```

> Konfigurationsdateien und Umgebungsvariablen können Geheimnisse enthalten. Nicht wahllos nach Inhalten suchen oder vollständige Dateien in Tickets kopieren.

---

12. Aktive Netzwerkabhängigkeiten eines Prozesses erkennen

Bereits aufgebaute Verbindungen können Hinweise auf verwendete Backends liefern.

Windows:

```
[RO][SENS] Get-NetTCPConnection -OwningProcess <PID> |
    Select-Object LocalAddress,
                  LocalPort,
                  RemoteAddress,
                  RemotePort,
                  State
```

Linux:

```
[RO][SENS][PRIV] sudo lsof -nP -a -p <PID> -i
```

macOS:

```
[RO][SENS][PRIV] sudo lsof -nP -a -p <PID> -i
```

Damit können Hinweise gefunden werden auf:

- Datenbankserver,
- Authentifizierungsserver,
- SMTP-Server,
- Proxys,
- externe APIs,
- Clusterknoten,
- Objekt- oder Netzwerkspeicher,
- Monitoring- und Protokollserver.

Einschränkungen:

- Nur aktuell geöffnete Verbindungen sind sichtbar.
- Kurzlebige Verbindungen können zwischen zwei Abfragen fehlen.
- Ein Ziel kann über einen Proxy vermittelt werden.
- IP-Adressen allein beweisen nicht die fachliche Funktion des Zielsystems.
- UDP-Kommunikation ist schwieriger zu bewerten.
- Eine aktuell fehlende Verbindung kann bei Leerlauf normal sein.

---

13. DNS-Abhängigkeit prüfen

Wenn ein Dienst ein Backend über einen Namen anspricht, muss genau dieser Name aus dem Kontext des Dienstservers geprüft werden.

Aufgabe	Windows	Linux	macOS
A- und AAAA-Auflösung	`[TEST] Resolve-DnsName "<BACKEND>"`	`[TEST] getent ahosts "<BACKEND>"`	`[TEST] dscacheutil -q host -a name "<BACKEND>"`
DNS gezielt abfragen	`[TEST] Resolve-DnsName "<BACKEND>" -Server <DNS-IP>`	`[TEST] dig @<DNS-IP> "<BACKEND>" A`	`[TEST] dig @<DNS-IP> "<BACKEND>" A`
IPv6-Adresse abfragen	`[TEST] Resolve-DnsName "<BACKEND>" -Type AAAA`	`[TEST] dig "<BACKEND>" AAAA`	`[TEST] dig "<BACKEND>" AAAA`

Zu prüfen:

- Wird der richtige Name verwendet?
- Wird die erwartete IP-Adresse geliefert?
- Existieren unerwartete alte Adressen?
- Gibt es unterschiedliche Ergebnisse auf Client und Server?

- Wird ein Suchsuffix benötigt?
- Verwendet die Anwendung einen vollständig qualifizierten Namen?
- Bevorzugt die Anwendung IPv6?
- Ist der DNS-Server aus dem Dienstkontext erreichbar?
- Wurde ein DNS-Eintrag kürzlich geändert?

> Eine erfolgreiche DNS-Abfrage beweist nur die Namensauflösung. Der zurückgegebene Zielserver muss anschließend über das erforderliche Protokoll geprüft werden.

---

14. TCP-Abhängigkeit eines Backends prüfen

Windows:

```
[TEST] Test-NetConnection "<BACKEND>" -Port <PORT>
```

Linux:

```
[TEST] nc -vz -w 3 "<BACKEND>" <PORT>
```

macOS:

```
[TEST] nc -vz -w 3 "<BACKEND>" <PORT>
```

Mögliche Ergebnisse:

Ergebnis	Bedeutung
Namensauflösung fehlerhaft	DNS-Abhängigkeit gestört
Verbindung abgelehnt	Ziel erreichbar, aber kein passender Listener oder aktive Ablehnung
Zeitüberschreitung	Filterung, Routingproblem oder nicht reagierendes Ziel möglich
TCP-Verbindung erfolgreich	Transportweg bis zum Port funktioniert
Anwendung antwortet trotzdem fehlerhaft	Protokoll, TLS, Anmeldung oder Backendfunktion prüfen

> Der Test muss vom tatsächlichen Anwendungsserver oder aus einem technisch gleichwertigen Netzwerkpfad erfolgen. Ein erfolgreicher Test vom Administrator-Client beweist nicht, dass der Dienstserver denselben Zugriff besitzt.

---

15. HTTP- oder API-Abhängigkeit prüfen

Nur Antwortheader eines HTTP- oder HTTPS-Endpunkts anfordern:

Betriebssystem	Befehl
Windows	`[TEST][SENS] curl.exe -I --connect-timeout 5 "https://<BACKEND>/<PFAD>"`
Linux	`[TEST][SENS] curl -I --connect-timeout 5 "https://<BACKEND>/<PFAD>"`
macOS	`[TEST][SENS] curl -I --connect-timeout 5 "https://<BACKEND>/<PFAD>"`

Ausführliche TLS- und Verbindungsdiagnose:

Betriebssystem	Befehl
Windows	`[TEST][SENS] curl.exe -v --connect-timeout 5 "https://<BACKEND>/<PFAD>"`
Linux	`[TEST][SENS] curl -v --connect-timeout 5 "https://<BACKEND>/<PFAD>"`
macOS	`[TEST][SENS] curl -v --connect-timeout 5 "https://<BACKEND>/<PFAD>"`

Zu prüfen:

- Wird die erwartete IP-Adresse verwendet?
- Ist der TCP-Verbindungsaufbau erfolgreich?
- Funktioniert der TLS-Handshake?
- Passt der Zertifikatsname?
- Welcher HTTP-Statuscode wird geliefert?
- Antwortet der erwartete Server?
- Erfolgt eine unerwartete Weiterleitung?
- Ist eine Authentifizierung erforderlich?
- Wie lange dauert die Antwort?

> `-v` kann Header und andere sensible Informationen anzeigen. Zugangstoken oder Sitzungscookies dürfen nicht in die Befehlszeile oder Dokumentation übernommen werden.

---

16. Speicher- und Dateisystemabhängigkeiten prüfen

Aufgabe	Windows	Linux	macOS
Eingebundene Datenträger	`[RO] Get-Volume`	`[RO] findmnt`	`[RO] mount`
Freier Speicherplatz	`[RO] Get-Volume`	`[RO] df -hT`	`[RO] df -h`
Bestimmten Pfad prüfen	`[RO] Test-Path "<PFAD>"`	`[RO] findmnt --target "<PFAD>"`	`[RO] df -h "<PFAD>"`
Pfadinformationen	`[RO][FILE] Get-Item "<PFAD>"`	`[RO][FILE] stat "<PFAD>"`	`[RO][FILE] stat "<PFAD>"`
Schreibrechte des aktuellen Kontos	`[RO][FILE] Get-Acl "<PFAD>"`	`[RO][FILE] namei -l "<PFAD>"`	`[RO][FILE] ls -lde "<PFAD>"`

Zu prüfen:

- Ist das Dateisystem eingehängt?
- Zeigt der Pfad auf das erwartete Ziel?
- Ist ausreichend Speicherplatz vorhanden?
- Sind unter Linux noch Inodes verfügbar?
- Ist das Dateisystem schreibgeschützt?
- Besitzt das Dienstkonto die erforderlichen Rechte?
- Ist ein Netzwerkmount nur scheinbar vorhanden, aber nicht erreichbar?
- Bestehen Dateisperren?
- Hat sich der Mountpfad geändert?
- Ist das Volume nach einem Neustart automatisch eingebunden worden?

> Schreibtests verändern das Dateisystem und dürfen nur gezielt, mit freigegebenem Testpfad und anschließendem Aufräumen durchgeführt werden.

---

17. Zeit- und Synchronisationsabhängigkeit prüfen

Falsche Systemzeit kann unter anderem folgende Funktionen beeinträchtigen:

- Kerberos-Authentifizierung,
- Zertifikatsprüfung,
- signierte Tokens,
- Protokollkorrelation,
- zeitgesteuerte Aufgaben,
- Datenbankreplikation,
- Clusterentscheidungen,
- Ablaufzeiten und Cache-Gültigkeit.

Aufgabe	Windows	Linux	macOS
Lokale Zeit	`[RO] Get-Date -Format o`	`[RO] date --iso-8601=seconds`	`[RO] date "+%Y-%m-%dT%H:%M:%S%z"`
UTC-Zeit	`[RO] (Get-Date).ToUniversalTime().ToString("o")`	`[RO] date -u "+%Y-%m-%dT%H:%M:%SZ"`	`[RO] date -u "+%Y-%m-%dT%H:%M:%SZ"`
Zeitzone	`[RO] Get-TimeZone`	`[RO] timedatectl status`	`[RO][PRIV] sudo systemsetup -gettimezone`
Zeitstatus	`[RO] w32tm /query /status`	`[RO] timedatectl status`	`[RO] systemsetup -getusingnetworktime`

Prüfung:

- Stimmen Datum und Uhrzeit?
- Ist die richtige Zeitzone eingestellt?
- Ist die Zeitquelle erreichbar?
- Sind Client, Dienstserver und Backend ausreichend synchron?
- Stimmen Protokollzeitstempel tatsächlich überein?
- Verwendet eine Anwendung UTC, während eine andere lokale Zeit protokolliert?

---

18. Authentifizierungsabhängigkeit prüfen

Typische abhängige Identitätsdienste:

- Active Directory,
- LDAP oder LDAPS,
- Kerberos,
- RADIUS,
- lokale Benutzerverwaltung,
- OAuth- oder OpenID-Connect-Anbieter,
- SAML-Identitätsanbieter,
- Zertifikats- oder Smartcard-Infrastruktur.

Vor einer Anmeldeprüfung kontrollieren:

1. Wird der Identitätsserver korrekt aufgelöst?
2. Ist der erforderliche Port erreichbar?
3. Ist die Systemzeit synchron?
4. Ist das Dienstkonto aktiv?
5. Ist das Kennwort beziehungsweise Secret gültig?
6. Ist das Konto gesperrt?
7. Besitzt das Konto die erforderlichen Rechte?
8. Ist das Serverzertifikat gültig?
9. Funktioniert nur die Anmeldung nicht oder auch die gesamte Anwendung?
10. Sind alle Benutzer oder nur einzelne Konten betroffen?

> Kennwörter, Tokens und private Schlüssel dürfen niemals in Diagnoseausgaben, Tickets oder BookStack-Beispiele kopiert werden.

---

19. Gemeinsame Abhängigkeit als Ursache erkennen

Wenn mehrere Dienste gleichzeitig ausfallen, sollte nach einer gemeinsamen Komponente gesucht werden.

Beobachtung	Mögliche gemeinsame Abhängigkeit
Mehrere Anwendungen können Benutzer nicht anmelden	Verzeichnisdienst, DNS oder Zeitquelle
Mehrere Webseiten liefern Backendfehler	Datenbank, Reverse Proxy oder Speicher
Mehrere Dienste können keine Dateien schreiben	Volume, Dateisystem oder Berechtigung
Nur externe APIs schlagen fehl	Proxy, Internetzugang, DNS oder Zertifikatskette
Alle Dienste eines Hosts sind langsam	CPU, RAM, Datenträger oder Virtualisierungsplattform
Mehrere Container starten nicht	Container-Runtime, Netzwerk oder Storage
Anwendungen an einem Standort sind betroffen	WAN, VPN, DNS oder Firewall
Fehler beginnt exakt gleichzeitig	Gemeinsame Änderung oder Infrastrukturkomponente

Vorgehen:

1. Betroffene Dienste auflisten
2. Zeitpunkt jedes Ausfalls vergleichen

```
3. Gemeinsame DNS-, Netzwerk- und Speicherpfade bestimmen
4. Gemeinsame Datenbanken und Identitätsdienste bestimmen
5. Gemeinsame Änderungen ermitteln
6. Gemeinsame Abhängigkeit direkt prüfen
7. Nach Wiederherstellung alle abhängigen Funktionen testen
```

---

20. Startreihenfolge nicht mit Betriebsbereitschaft verwechseln

Ein Dienst kann laut Betriebssystem gestartet worden sein, obwohl seine Funktion noch nicht vollständig bereitsteht.

Beispiel:

```
00:00 Datenbankprozess startet
00:02 Dienstverwaltung meldet „läuft“
00:05 Datenbank führt Wiederherstellung aus
00:40 Datenbank nimmt Anmeldungen an
```

Startet die Anwendung bereits bei Sekunde 3, kann ihre Verbindung fehlschlagen, obwohl der Datenbankdienst formal läuft.

Mögliche Lösungen innerhalb eines geplanten Änderungsverfahrens:

- geeignete Bereitschaftsprüfung,
- Wiederholungsversuche mit Begrenzung,
- zunehmende Wartezeiten zwischen Versuchen,
- sinnvoller Verbindungstimeout,
- korrekte systemd-Abhängigkeit,
- orchestrierte Healthchecks,
- Start erst nach erfolgreichem Backendtest,
- Warteschlange für vorübergehend nicht verarbeitbare Aufgaben.

> Eine feste Wartezeit ist häufig weniger zuverlässig als eine echte Bereitschaftsprüfung.

---

21. Auswirkungen vor einem Neustart einer Abhängigkeit prüfen

Vor dem Neustart eines gemeinsam verwendeten Dienstes müssen die abhängigen Verbraucher bestimmt werden.

Windows:

```
[RO] (Get-Service -Name "<DIENSTNAME>").DependentServices |
    Select-Object Name, DisplayName, Status
```

Linux:

```
[RO] systemctl list-dependencies \
  --reverse \
  --no-pager \
  "<DIENST>"
```

macOS:

Da launchd kein allgemeines ausdrückliches Abhängigkeitsmodell besitzt, müssen zusätzlich geprüft werden:

- Herstellerdokumentation,
- aktive Netzwerkverbindungen,
- Konfigurationsdateien,
- Architekturübersicht,
- Monitoring,
- Anwendungsprotokolle,
- bekannte Verbraucher des Dienstes.

Freigabecheck:

```
[ ] Alle bekannten Verbraucher identifiziert
[ ] Aktive Sitzungen und Transaktionen geprüft
[ ] Cluster- oder Failover-Verhalten geprüft
[ ] Wiederverbindungsverhalten der Clients bekannt
[ ] Neustartreihenfolge festgelegt
[ ] Wartungsfenster beziehungsweise Freigabe vorhanden
[ ] Diagnoseinformationen vorab gesichert
[ ] Funktionstests für alle wichtigen Verbraucher vorbereitet
```

```
[ ] Rückfallmöglichkeit festgelegt
```

22. Abhängigkeitsfehler schrittweise eingrenzen

Empfohlene Reihenfolge:

1. Hauptfunktion und Fehlermeldung bestimmen.
2. Formale Dienstabhängigkeiten erfassen.
3. Architektur und funktionale Abhängigkeiten ergänzen.
4. Kritische und optionale Abhängigkeiten kennzeichnen.
5. Namen der Zielsysteme prüfen.
6. DNS-Auflösung prüfen.
7. Netzwerkport vom tatsächlichen Dienstserver prüfen.
8. TLS- und Zertifikatsprüfung durchführen.
9. Protokollspezifischen Funktionstest durchführen.
10. Authentifizierung mit einem freigegebenen Testkonto prüfen.
11. Speicher- und Ressourcenabhängigkeiten prüfen.
12. Zeitstempel der beteiligten Systeme vergleichen.
13. Protokolle auf Quelle und Ziel korrelieren.
14. Gemeinsame Abhängigkeiten anderer gestörter Dienste prüfen.
15. Erst nach gesicherter Ursache eine Änderung planen.

Prüfkette:

```
Name korrekt?
  ↓
DNS korrekt?
  ↓
Route vorhanden?
  ↓
Port erreichbar?
  ↓
TLS gültig?
  ↓
Protokoll antwortet?
  ↓
Authentifizierung erfolgreich?
  ↓
Berechtigung ausreichend?
```

↓

Fachliche Anfrage erfolgreich?

23. Typische Fehlinterpretationen

Fehlinterpretation	Richtige Bewertung
„Alle Abhängigkeiten stehen in der Dienstverwaltung.“	Viele funktionale Abhängigkeiten sind nur in der Anwendung konfiguriert
„Der erforderliche Dienst läuft, also funktioniert er.“	Seine fachliche Funktion muss separat geprüft werden
„`After=` bedeutet, dass der andere systemd-Dienst zwingend benötigt wird.“	`After=` legt nur eine Reihenfolge fest
„`Requires=` bedeutet automatisch die richtige Startreihenfolge.“	Dafür kann zusätzlich `After=` oder `Before=` erforderlich sein
„Ein erfolgreicher Porttest beweist die Backendfunktion.“	Nur der Transportweg wurde bestätigt
„DNS liefert eine Adresse, also ist das richtige Backend erreicht.“	Adresse, Zertifikat und Anwendungsantwort müssen geprüft werden
„Alle Benutzer sind betroffen, also liegt es am Hauptdienst.“	Eine gemeinsame Abhängigkeit kann ausgefallen sein
„Ein Neustart der Hauptanwendung repariert die Abhängigkeit.“	Er kann nur neue Verbindungsversuche auslösen
„Startreihenfolge beweist Betriebsbereitschaft.“	Ein Prozess kann laufen, während die Initialisierung noch andauert
„Ein optionales Backend kann ignoriert werden.“	Es kann für einzelne wichtige Funktionen zwingend sein

24. Checkliste zur Abhängigkeitsanalyse

[ ] Hauptdienst eindeutig bestimmt

[ ] Formale Betriebssystem-Abhängigkeiten erfasst

[ ] Vorwärts gerichtete Abhängigkeiten geprüft

[ ] Rückwärts gerichtete Abhängigkeiten geprüft

[ ] Startreihenfolge getrennt von Anforderung bewertet

[ ] Kritische und optionale Abhängigkeiten unterschieden

[ ] Externe Datenbanken erfasst

[ ] DNS- und Zeitabhängigkeiten erfasst
[ ] Authentifizierungsdienste erfasst
[ ] Speicher- und Dateisystemabhängigkeiten erfasst
[ ] Proxys, Loadbalancer und APIs erfasst
[ ] Container-, Cluster- oder Cloud-Abhängigkeiten berücksichtigt
[ ] DNS-Auflösung vom Dienstserver geprüft
[ ] Zielport vom Dienstserver geprüft
[ ] TLS und Zertifikatsname geprüft
[ ] Protokollspezifischer Funktionstest durchgeführt
[ ] Gemeinsame Abhängigkeiten anderer Störungen geprüft
[ ] Protokollzeitstempel korreliert
[ ] Auswirkungen eines Neustarts bewertet
[ ] Keine Geheimnisse dokumentiert
[ ] Nächsten Diagnoseschritt festgelegt

Bewertung des Ergebnisses

Ergebnis	Nächster Schritt
Formale Abhängigkeit ist beendet	Status, Startfehler und Protokolle dieser Abhängigkeit prüfen
Abhängigkeit ist deaktiviert oder maskiert	Sollzustand und Änderungshistorie klären
DNS-Auflösung schlägt fehl	DNS-Konfiguration und verwendeten Namen prüfen
Port der Abhängigkeit ist nicht erreichbar	Listener, Firewall und Netzwerkpfad prüfen
Port ist erreichbar, Protokolltest scheitert	TLS, Authentifizierung und Anwendung prüfen
Speicherpfad fehlt	Mount, Volume, Netzwerk und Berechtigungen prüfen
Nur Anmeldung schlägt fehl	Identitätsdienst, Zeit und Dienstkonto prüfen
Mehrere Dienste sind gleichzeitig betroffen	Gemeinsame Abhängigkeit priorisiert untersuchen
Backend ist erst spät betriebsbereit	Healthcheck, Wiederholungslogik und Startablauf prüfen
Alle Abhängigkeiten funktionieren	Hauptdienstkonfiguration und Dienstprotokolle untersuchen

---

Merksatz

> **Die sichtbare Anwendung ist oft nur das letzte Glied einer langen Dienstkette. Die Ursache liegt häufig in einer gemeinsam genutzten,**

> **zunächst unsichtbaren Abhängigkeit.**

Weiterführende Quellen

- Microsoft Learn – Get-Service
- Microsoft Learn – ServiceController.ServicesDependedOn
- Microsoft Learn – ServiceController.DependentServices
- Microsoft Learn – sc.exe qc
- Microsoft Learn – sc.exe enumdepend
- systemd – systemctl
- systemd – systemd.unit
- systemd – systemd.service
- Apple – launchctl-Handbuchseite
- Apple – launchd.plist-Handbuchseite
- Microsoft Learn – Test-NetConnection
- curl – offizielle Dokumentation
