4.7 Dienststartfehler analysieren

Ein Dienststartfehler liegt vor, wenn ein Dienst nicht gestartet werden kann, im Startvorgang hängen bleibt oder unmittelbar nach dem Start wieder beendet wird.

> **Grundsatz:**
> Nicht mehrfach auf „Starten“ klicken. Zuerst Status, Exitcode, Protokolle und Startkonfiguration sichern, damit die ursprüngliche Fehlerursache nicht durch Folgeereignisse verdeckt wird.

Ziele dieser Seite

Nach dieser Seite sollst du:

- verschiedene Arten von Dienststartfehlern unterscheiden können,
- Exitcodes und Beendigungsursachen erfassen können,
- Programmpfad und Startparameter prüfen können,
- Fehler des Dienstkontos und der Berechtigungen erkennen können,
- fehlende Abhängigkeiten und Ressourcen identifizieren können,
- Zeitüberschreitungen und sofortige Prozessabbrüche unterscheiden können,
- Startbegrenzungen und Neustartschleifen erkennen können,
- Windows-, systemd- und launchd-Startfehler analysieren können,
- einen kontrollierten Startversuch vorbereiten und auswerten können.

---

1. Arten von Dienststartfehlern unterscheiden

Fehlerbild	Beschreibung	Mögliche Ursache
Dienst nicht gefunden	Dienstverwaltung kennt den Namen nicht	Falscher Name, Dienst nicht installiert oder andere Umgebung
Dienst deaktiviert	Start wird durch Konfiguration verhindert	Bewusste Deaktivierung oder fehlerhafte Änderung
Dienst maskiert	systemd blockiert den Start vollständig	Administrativer Schutz oder fehlerhafte Maskierung
Programmdatei fehlt	Angegebene ausführbare Datei ist nicht vorhanden	Unvollständiges Update, Löschung oder falscher Pfad
Zugriff verweigert	Dienst darf Programm, Datei oder Ressource nicht verwenden	Konto- oder Berechtigungsfehler

Fehlerbild	Beschreibung	Mögliche Ursache
Abhängigkeit fehlt	Benötigter Dienst oder Backend ist nicht verfügbar	Dienst-, DNS-, Netzwerk- oder Speicherfehler
Start dauert zu lange	Dienst meldet nicht rechtzeitig Betriebsbereitschaft	Blockierte Initialisierung oder langsames Backend
Prozess beendet sich sofort	Programm startet und bricht unmittelbar ab	Konfiguration, Portkonflikt oder fehlende Ressource
Prozess stürzt ab	Unbehandelte Ausnahme oder schwerer Laufzeitfehler	Softwarefehler, Bibliothek oder inkompatible Version
Neustartschleife	Dienst wird automatisch fortlaufend neu gestartet	Wiederherstellungsregel und unveränderte Fehlerursache
Startgrenze erreicht	Dienstverwaltung unterbindet weitere Versuche	Zu viele Fehlstarts innerhalb kurzer Zeit
Dienst läuft, ist aber nicht bereit	Prozess existiert, Initialisierung ist unvollständig	Backend, Migration, Cache oder Healthcheck

2. Mindestinformationen vor einem neuen Startversuch sichern

- [ ] Datum, Uhrzeit und Zeitzone
- [ ] Server und Betriebssystem
- [ ] Interner Dienstname beziehungsweise launchd-Label
- [ ] Aktueller Dienststatus
- [ ] Startart beziehungsweise Aktivierungszustand
- [ ] Prozess-ID, falls vorhanden
- [ ] Letzter Exitcode
- [ ] Dienstspezifischer Exitcode
- [ ] Programmpfad und Startparameter
- [ ] Dienstkonto
- [ ] Formale Abhängigkeiten
- [ ] Wiederherstellungs- und Neustartregeln
- [ ] Relevante Systemprotokolle
- [ ] Relevante Anwendungsprotokolle
- [ ] Letzte Konfigurations- oder Softwareänderung

> Erst nach dieser Sicherung sollte ein weiterer kontrollierter Startversuch durchgeführt werden.

---

3. Allgemeiner Diagnoseweg bei Startfehlern

```
Dienst vorhanden?
  ↓
Aktiviert und nicht blockiert?
  ↓
Programmpfad vorhanden?
  ↓
Startkonfiguration syntaktisch gültig?
  ↓
Dienstkonto gültig?
  ↓
Datei- und Verzeichnisrechte ausreichend?
  ↓
Abhängigkeiten verfügbar?
  ↓
Port und andere Ressourcen verfügbar?
  ↓
Prozess startet?
  ↓
Prozess bleibt aktiv?
  ↓
Dienst meldet Betriebsbereitschaft?
  ↓
Anwendungstest erfolgreich?
```

Empfohlene Reihenfolge:

1. Exakten Dienstnamen bestätigen.
2. Aktuellen Status erfassen.
3. Exitcode und Protokolle sichern.
4. Startart und Blockierungen prüfen.
5. Programmpfad und Argumente kontrollieren.

6. Dienstkonto und Berechtigungen prüfen.
7. Abhängigkeiten testen.
8. Portkonflikte und Ressourcen prüfen.
9. Konfiguration mit Herstellerwerkzeug validieren.
10. Einen einzelnen kontrollierten Startversuch durchführen.
11. Protokolle während dieses Versuchs beobachten.
12. Ergebnis dokumentieren und erneut bewerten.

---

4. Windows-Dienststatus und Exitcodes erfassen

Grundlegenden Dienststatus anzeigen:

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

Erweiterte Dienstinformationen anzeigen:

```
[RO] Get-CimInstance Win32_Service -Filter "Name='<DIENSTNAME>'" |
    Select-Object Name,
                  DisplayName,
                  State,
                  Status,
                  StartMode,
                  StartName,
                  ProcessId,
                  ExitCode,
                  ServiceSpecificExitCode
```

Status mit `sc.exe` prüfen:

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

Wichtige Angaben:

Angabe	Bedeutung
`STATE`	Aktueller Dienstzustand
`WIN32_EXIT_CODE`	Windows- beziehungsweise Dienstausgangscode

Angabe	Bedeutung
`SERVICE_EXIT_CODE`	Dienstspezifischer Ausgangscode
`CHECKPOINT`	Fortschrittswert bei längeren Zustandswechseln
`WAIT_HINT`	Vom Dienst geschätzte Wartezeit
`PID`	Aktuelle Prozess-ID, sofern vorhanden

> Ein Exitcode `0` ist nur dann aussagekräftig, wenn der Dienst regulär beendet wurde. Ein Dienst kann trotzdem seine vorgesehene Funktion nicht bereitgestellt haben.

---

5. Windows-Startkonfiguration prüfen

Konfiguration des Dienstes anzeigen:

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

Konfiguration über CIM anzeigen:

```
[RO][SENS] Get-CimInstance Win32_Service -Filter "Name='<DIENSTNAME>'" |
    Select-Object Name,
                  StartMode,
                  StartName,
                  PathName,
                  ServiceType,
                  DesktopInteract
```

Zu prüfen sind:

Feld	Prüffrage
`BINARY_PATH_NAME` beziehungsweise `PathName`	Ist der Programmpfad korrekt?
Startparameter	Sind Argumente vollständig und korrekt maskiert?
`SERVICE_START_NAME` beziehungsweise `StartName`	Wird das vorgesehene Dienstkonto verwendet?
`START_TYPE` beziehungsweise `StartMode`	Ist der Dienst deaktiviert?

Feld	Prüffrage
`DEPENDENCIES`	Sind alle formalen Abhängigkeiten vorhanden?
`TYPE` beziehungsweise `ServiceType`	Passt die Dienstart zur Anwendung?

Wichtig bei Programmpfaden:

- Programmpfad und Argumente dürfen nicht verwechselt werden.
- Leerzeichen im Pfad erfordern korrekte Anführungszeichen.
- Relative Pfade können im Dienstkontext anders aufgelöst werden.
- Ein Netzlaufwerksbuchstabe aus einer Benutzersitzung steht einem Dienst normalerweise nicht automatisch zur Verfügung.
- Das Dienstkonto muss auf Programm, Arbeitsverzeichnis und Konfiguration zugreifen können.

---

6. Windows-Programmpfad kontrolliert prüfen

Da `PathName` auch Argumente enthalten kann, darf die gesamte Zeichenfolge nicht ungeprüft an `Test-Path` übergeben werden.

Konfigurierte Zeichenfolge zunächst nur anzeigen:

```
[RO][SENS] Get-CimInstance Win32_Service -Filter "Name='<DIENSTNAME>'" |
    Select-Object Name, PathName
```

Anschließend wird der tatsächliche ausführbare Pfad aus der Ausgabe ermittelt und separat geprüft:

```
[RO][FILE] Test-Path -LiteralPath "<AUSFÜHRBARE_DATEI>"
```

Dateiinformationen anzeigen:

```
[RO][FILE][SENS] Get-Item -LiteralPath "<AUSFÜHRBARE_DATEI>" |
    Select-Object FullName, Length, CreationTime, LastWriteTime, VersionInfo
```

Berechtigungen anzeigen:

```
[RO][FILE][SENS] Get-Acl -LiteralPath "<AUSFÜHRBARE_DATEI>" |
    Format-List
```

Zusätzlich prüfen:

- Ist die Datei tatsächlich vorhanden?
- Wurde sie kürzlich ersetzt oder verschoben?
- Passt die Architektur zum Betriebssystem?
- Sind benötigte Bibliotheken vorhanden?
- Blockiert Sicherheitssoftware die Ausführung?
- Liegt die Datei auf einem beim Start noch nicht verfügbaren Datenträger?
- Ist der Pfad lokal oder von einer Netzwerkressource abhängig?

---

7. Windows-Dienstkonto und Anmelderechte prüfen

Dienstkonto anzeigen:

```
[RO] Get-CimInstance Win32_Service -Filter "Name='<DIENSTNAME>'" |
    Select-Object Name, StartName, State, StartMode
```

Mögliche Kontotypen:

- `LocalSystem`,
- `NT AUTHORITY\LocalService`,
- `NT AUTHORITY\NetworkService`,
- virtuelles Dienstkonto,
- verwaltetes Dienstkonto,
- lokales Benutzerkonto,
- Domänenkonto.

Zu prüfen:

- Existiert das Konto?
- Ist es aktiviert oder gesperrt?
- Ist ein hinterlegtes Kennwort abgelaufen oder geändert worden?
- Besitzt es das Recht „Anmelden als Dienst“?
- Darf es die Programmdatei lesen und ausführen?
- Darf es Konfigurations- und Datenverzeichnisse verwenden?
- Darf es Zertifikate und private Schlüssel verwenden?
- Darf es auf Netzwerkressourcen zugreifen?

- Wurde das Konto kürzlich geändert?
- Funktioniert ein verwaltetes Dienstkonto ordnungsgemäß?

> Ein manueller Programmstart als Administrator beweist nicht, dass das Programm unter dem tatsächlichen Dienstkonto funktioniert.

8. Typische Windows-Startfehler einordnen

Fehler	Typische Bedeutung	Prüfrichtung
Fehler 2	Angegebene Datei wurde nicht gefunden	Programmpfad, Bibliothek oder Konfigurationspfad
Fehler 5	Zugriff verweigert	Konto, Rechte oder Sicherheitssoftware
Fehler 1068	Abhängigkeitsdienst oder Abhängigkeitsgruppe konnte nicht gestartet werden	Formale Abhängigkeiten
Fehler 1069	Anmeldung des Dienstkontos fehlgeschlagen	Konto, Kennwort und Anmelderecht
Fehler 1053	Dienst antwortete nicht rechtzeitig auf Start- oder Steuerungsanforderung	Initialisierung, Timeout, Programmfehler
Fehler 1067	Prozess wurde unerwartet beendet	Anwendungsprotokoll, Konfiguration oder Absturz
Fehler 1079	Konto stimmt nicht mit anderen Diensten im gemeinsamen Prozess überein	Dienstkonto und gemeinsam verwendeter Prozess
Fehler 193	Keine gültige Win32-Anwendung	Architektur, Dateityp oder fehlerhafte Programmdatei

> Die vollständige Fehlermeldung, das betroffene Produkt und die Windows-Version müssen immer mit dokumentiert werden. Die Tabelle ist eine Eingrenzungshilfe, kein Ersatz für die konkrete Herstellerdiagnose.

9. Windows-Ereignisse eines Startversuchs auswerten

Service-Control-Manager-Ereignisse der letzten Stunde:

```
[RO][SENS] Get-WinEvent -FilterHashtable @{
    LogName      = "System"
    ProviderName = "Service Control Manager"
    StartTime    = (Get-Date).AddHours(-1)
} |
    Select-Object TimeCreated,
                  Id,
                  LevelDisplayName,
                  Message
```

Typische startbezogene Ereignisse filtern:

```
[RO][SENS] Get-WinEvent -FilterHashtable @{
    LogName      = "System"
    ProviderName = "Service Control Manager"
    Id           = 7000, 7001, 7009, 7011, 7023, 7024, 7031, 7034
    StartTime    = (Get-Date).AddHours(-24)
} |
    Select-Object TimeCreated, Id, LevelDisplayName, Message
```

System- und Anwendungsereignisse im festen Zeitfenster verbinden:

```
[RO][SENS] $startTime = Get-Date "2026-07-31 09:10:00"
$endTime   = Get-Date "2026-07-31 09:20:00"

Get-WinEvent -FilterHashtable @{
    LogName   = "System", "Application"
    StartTime = $startTime
    EndTime   = $endTime
} |
    Sort-Object TimeCreated |
    Select-Object TimeCreated,
                  LogName,
                  ProviderName,
                  Id,
```

```
    LevelDisplayName,
    Message
```

---

10. Windows-Wiederherstellungsaktionen prüfen

Ein Dienst kann nach einem Fehlstart oder Absturz automatisch neu gestartet werden.

Konfigurierte Fehleraktionen anzeigen:

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

Zusätzlichen Fehleraktionsstatus anzeigen:

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

Mögliche Aktionen:

- keine Aktion,
- Dienst neu starten,
- Programm ausführen,
- Computer neu starten.

Prüffragen:

- Wird der Dienst automatisch neu gestartet?
- Nach welcher Wartezeit?
- Wie viele Versuche erfolgen?
- Wann wird der Fehlerzähler zurückgesetzt?
- Wird auch bei einem normalen Prozessende eine Fehleraktion ausgelöst?
- Verändert die Wiederherstellung die sichtbare PID?
- Verdeckt die Automatik einen wiederkehrenden Startfehler?

> Wiederherstellungsregeln dürfen nicht ungeprüft verändert werden. Sie können die Verfügbarkeit und das Verhalten bei Abstürzen erheblich beeinflussen.

---

11. systemd-Status und Startursache unter Linux prüfen

Ausführlichen Status anzeigen:

```
[RO][SENS][PRIV] sudo systemctl status "<DIENST>" --no-pager
```

Wichtige Zustände strukturiert anzeigen:

```
[RO] systemctl show "<DIENST>" \
  --
property=LoadState,ActiveState,SubState,UnitFileState,Result,MainPID,ExecMainCode,ExecMainStatus,Stat
usErrno,NRestarts
```

Prüfen, ob die Unit aktiviert ist:

```
[RO] systemctl is-enabled "<DIENST>"
```

Prüfen, ob sie fehlgeschlagen ist:

```
[RO] systemctl is-failed "<DIENST>"
```

Alle fehlgeschlagenen Units anzeigen:

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

Wichtige Angaben:

Eigenschaft	Bedeutung
`LoadState`	Wurde die Unit korrekt geladen?
`ActiveState`	Übergeordneter Aktivzustand
`SubState`	Detaillierter Unterzustand
`Result`	Ergebnis des letzten Start- oder Laufvorgangs
`MainPID`	PID des Hauptprozesses
`ExecMainCode`	Art der Prozessbeendigung
`ExecMainStatus`	Exitcode oder Signalnummer

Eigenschaft	Bedeutung
`StatusErrno`	Gemeldeter Fehlerwert, sofern vorhanden
`NRestarts`	Anzahl automatischer Neustarts

12. systemd-Ergebniswerte interpretieren

Mögliche `Result`-Werte können unter anderem sein:

Ergebnis	Typische Bedeutung
`success`	Vorgang wurde aus Sicht von systemd erfolgreich beendet
`exit-code`	Prozess beendete sich mit nicht als erfolgreich bewertetem Exitcode
`signal`	Prozess wurde durch ein Signal beendet
`core-dump`	Prozess wurde durch Signal beendet und erzeugte einen Core Dump
`timeout`	Start-, Stopp- oder Laufzeitgrenze wurde überschritten
`watchdog`	Watchdog-Reaktion blieb aus
`start-limit-hit`	Zu viele Startversuche in kurzer Zeit
`resources`	Benötigte Systemressource konnte nicht bereitgestellt werden
`protocol`	Dienst erfüllte das erwartete Startprotokoll nicht
`dependency`	Eine benötigte Unit schlug fehl
`exec-condition`	Eine konfigurierte Startbedingung verhinderte den Start

Beendigungsart:

`ExecMainCode`	Grundbedeutung
`exited`	Prozess beendete sich über einen Exitcode
`killed`	Prozess wurde durch ein Signal beendet
`dumped`	Prozess wurde durch Signal beendet und erzeugte möglicherweise einen Dump

> Die konkrete Bedeutung von `ExecMainStatus` hängt davon ab, ob der Prozess regulär beendet oder durch ein Signal beendet wurde. Zusätzlich muss die Dokumentation des Programms geprüft werden.

---

13. systemd-Unit und wirksame Startparameter prüfen

Wirksame Unit einschließlich Drop-ins anzeigen:

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

Wichtige Startparameter strukturiert anzeigen:

```
[RO][SENS] systemctl show "<DIENST>" \
  --
property=Type,User,Group,ExecStart,ExecStartPre,ExecStartPost,WorkingDirectory,EnvironmentFiles,Frag
mentPath,DropInPaths
```

Unit-Datei formal prüfen:

```
[RO][FILE] systemd-analyze verify "<UNIT-DATEI>"
```

Zu prüfen sind:

- `Type=`,
- `ExecStart=`,
- `ExecStartPre=`,
- `ExecStartPost=`,
- `User=` und `Group=`,
- `WorkingDirectory=`,
- `EnvironmentFile=`,
- `PIDFile=`,
- `RuntimeDirectory=`,
- `StateDirectory=`,
- `TimeoutStartSec=`,
- `Restart=`,
- `RestartSec=`,
- `StartLimitIntervalSec=`,

- `StartLimitBurst=`.

> `systemd-analyze verify` benötigt den tatsächlichen Pfad einer Unit-Datei. Der Pfad sollte mit `systemctl show <DIENST> --property=FragmentPath` ermittelt werden.

---

14. Linux-Programmpfad und Berechtigungen prüfen

Konfigurierten Startbefehl anzeigen:

```
[RO][SENS] systemctl show "<DIENST>" --property=ExecStart
```

Datei des bekannten Programmpfads prüfen:

```
[RO][FILE] stat "<AUSFÜHRBARE_DATEI>"
```

Dateityp und Architektur prüfen:

```
[RO][FILE] file "<AUSFÜHRBARE_DATEI>"
```

Berechtigungen aller Pfadbestandteile anzeigen:

```
[RO][FILE] namei -l "<AUSFÜHRBARE_DATEI>"
```

Dynamische Bibliotheksabhängigkeiten eines geeigneten dynamischen ELF-Programms anzeigen:

```
[RO][FILE][SENS] ldd "<AUSFÜHRBARE_DATEI>"
```

Wichtiger Sicherheitshinweis:

`ldd` sollte nicht ungeprüft auf nicht vertrauenswürdige ausführbare Dateien angewendet werden. Abhängig von System und Binärdatei kann die Auswertung Sicherheitsrisiken besitzen.

Alternativ kann bei ELF-Dateien zunächst statisch geprüft werden:

```
[RO][FILE] readelf -d "<AUSFÜHRBARE_DATEI>"
```

Zu prüfen:

- Datei vorhanden,
- ausführbar,
- richtige Architektur,
- korrekter Interpreter,
- benötigte Bibliotheken vorhanden,
- Dienstkonto kann alle Pfadbestandteile durchlaufen,
- Arbeitsverzeichnis vorhanden,
- Dateisystem nicht schreibgeschützt,
- Sicherheitsmechanismen wie SELinux oder AppArmor berücksichtigt.

---

15. Linux-Dienstkonto und Dienstkontext prüfen

Konfiguriertes Konto anzeigen:

```
[RO] systemctl show "<DIENST>" --property=User,Group,DynamicUser
```

Benutzerkonto prüfen:

```
[RO] getent passwd "<DIENSTBENUTZER>"
```

Gruppenmitgliedschaften anzeigen:

```
[RO] id "<DIENSTBENUTZER>"
```

Arbeitsverzeichnis prüfen:

```
[RO] systemctl show "<DIENST>" --property=WorkingDirectory
```

Referenzierte Umgebungsdateien anzeigen:

```
[RO][SENS] systemctl show "<DIENST>" --property=EnvironmentFiles
```

Zu prüfen:

- Existiert der Dienstbenutzer?
- Existiert die konfigurierte Gruppe?
- Sind Dateirechte ausreichend?
- Ist ein dynamischer Benutzer vorgesehen?
- Existiert das Arbeitsverzeichnis?
- Kann das Konto Konfigurations- und Datendateien lesen?
- Kann es in erforderliche Verzeichnisse schreiben?
- Kann es privilegierte Ports oder Geräte verwenden?
- Blockieren SELinux, AppArmor oder systemd-Sandboxing den Zugriff?
- Sind Secrets für den Dienstkontext verfügbar?

> Das Starten des Programms als `root` ist kein gültiger Nachweis für die Funktion unter dem tatsächlichen Dienstkonto.

---

16. systemd-Protokolle eines Startfehlers auswerten

Protokolle des aktuellen Systemstarts:

```
[RO][SENS][PRIV] sudo journalctl \
  -b \
  -u "<DIENST>" \
  --no-pager
```

Letzte 100 Einträge:

```
[RO][SENS][PRIV] sudo journalctl \
  -u "<DIENST>" \
  -n 100 \
  --output=short-iso \
  --no-pager
```

Fehler und schwerwiegendere Meldungen:

```
[RO][SENS][PRIV] sudo journalctl \
  -u "<DIENST>" \
  -p err \
  --since "1 hour ago" \
  --no-pager
```

Dienst- und Systemmeldungen im selben Zeitfenster:

```
[RO][SENS][PRIV] sudo journalctl \
  --since "2026-07-31 09:10:00" \
  --until "2026-07-31 09:20:00" \
  --output=short-iso \
  --no-pager
```

Kernelmeldungen berücksichtigen:

```
[RO][SENS][PRIV] sudo journalctl \
  -k \
  -b \
  -p warning \
  --no-pager
```

---

17. systemd-Neustartschleife und Startbegrenzung erkennen

Neustartregeln anzeigen:

```
[RO] systemctl show "<DIENST>" \
  --property=Restart,RestartUSec,NRestarts
```

Startbegrenzung anzeigen:

```
[RO] systemctl show "<DIENST>" \
  --property=StartLimitIntervalUSec,StartLimitBurst,Result
```

Anzeichen:

- `NRestarts` steigt,
- PID wechselt fortlaufend,
- Prozesslaufzeit bleibt sehr kurz,
- `Result=start-limit-hit`,
- Journal zeigt wiederholte identische Startfehler,
- Dienst ist zwischen den Versuchen kurz erreichbar,
- Healthcheck schlägt fortlaufend fehl.

Fehlerstatus erst nach der Beweissicherung zurücksetzen:

```
[CHANGE][PRIV] sudo systemctl reset-failed "<DIENST>"
```

`reset-failed`:

- startet den Dienst nicht,
- löscht aber den gespeicherten Fehlerzustand,
- setzt bestimmte Zähler für Startbegrenzungen zurück,
- verändert damit Diagnosezustand.

> Dieser Befehl darf deshalb erst nach Dokumentation von `Result`, `NRestarts`, Status und Journal verwendet werden.

---

18. systemd-Diensttypen als mögliche Fehlerursache prüfen

`Type=`	Erwartetes Verhalten
`simple`	Gestarteter Prozess gilt unmittelbar als Hauptprozess
`exec`	Start gilt nach erfolgreichem Ausführen des Programms als erfolgt
`forking`	Programm erzeugt Hintergrundprozess; häufig mit PID-Datei
`oneshot`	Prozess führt Aufgabe aus und beendet sich wieder
`notify`	Dienst meldet systemd ausdrücklich Betriebsbereitschaft
`dbus`	Bereitschaft hängt von einem D-Bus-Namen ab
`idle`	Ausführung wird verzögert, bis andere Jobs abgearbeitet sind

Mögliche Fehlkonfigurationen:

- `forking` verwendet falsche oder fehlende PID-Datei,
- `notify`-Dienst sendet keine Bereitschaftsmeldung,
- `oneshot` wird fälschlich als dauerhaft laufender Prozess erwartet,
- Programm forkt, obwohl `Type=simple` ungeeignet konfiguriert wurde,
- Wrapperprozess beendet sich und systemd überwacht nicht den erwarteten Prozess,
- Startbefehl blockiert und erreicht den vorgesehenen Zustand nicht.

> Der korrekte Diensttyp muss anhand der Programmdokumentation und des tatsächlichen Prozessverhaltens bestimmt werden.

---

19. launchd-Status und letzten Exitstatus unter macOS prüfen

Systemweiten Dienst prüfen:

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

Benutzerbezogenen Dienst prüfen:

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

Relevante Angaben können sein:

```
state
pid
runs
last exit code
program
arguments
reason
path
```

Deaktivierungsüberschreibungen prüfen:

```
[RO] launchctl print-disabled system
```

GUI-Domain des aktuellen Benutzers prüfen:

```
[RO] launchctl print-disabled "gui/$(id -u)"
```

Wichtig:

- Fehlende PID kann bei einem bedarfsgesteuerten Job normal sein.
- Ein von null abweichender letzter Exitstatus ist ein Diagnosehinweis.
- Eine steigende Anzahl unter `runs` kann auf wiederholte Startversuche hinweisen.
- Ein Job mit `KeepAlive` kann nach einem Fehler automatisch erneut gestartet werden.
- Häufige schnelle Fehlstarts können von launchd gedrosselt werden.

---

20. launchd-Konfiguration unter macOS prüfen

Bekannte systemweite plist anzeigen:

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

Formale plist-Prüfung durchführen:

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

Benutzerbezogene plist prüfen:

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

Zu prüfen sind:

Schlüssel	Prüffrage
`Label`	Stimmt das Label mit dem verwendeten Dienstziel überein?
`Program`	Existiert das Programm?
`ProgramArguments`	Sind Programm und Argumente korrekt angeordnet?

Schlüssel	Prüffrage
`WorkingDirectory`	Existiert das Arbeitsverzeichnis?
`UserName`	Existiert das Konto und besitzt es die nötigen Rechte?
`GroupName`	Existiert die Gruppe?
`KeepAlive`	Wird ein fehlerhafter Prozess wiederholt gestartet?
`RunAtLoad`	Soll der Job beim Laden gestartet werden?
`StandardOutPath`	Kann die Datei beziehungsweise das Verzeichnis verwendet werden?
`StandardErrorPath`	Kann die Fehlerausgabe geschrieben werden?
`EnvironmentVariables`	Sind erforderliche Variablen vorhanden?

> `ProgramArguments` ist ein Array. Das erste Element bezeichnet üblicherweise das auszuführende Programm, wenn nicht zusätzlich `Program` angegeben wurde.

---

21. macOS-Programmpfad und Berechtigungen prüfen

Programmdatei prüfen:

```
[RO][FILE] stat "<AUSFÜHRBARE_DATEI>"
```

Dateityp und Architektur anzeigen:

```
[RO][FILE] file "<AUSFÜHRBARE_DATEI>"
```

Berechtigungen und erweiterte Zugriffslisten anzeigen:

```
[RO][FILE][SENS] ls -lde "<AUSFÜHRBARE_DATEI>"
```

Code-Signatur anzeigen und prüfen:

```
[RO][FILE][SENS] codesign \
  --verify \
  --deep \
  --strict \
  --verbose=2 \
  "<AUSFÜHRBARE_DATEI>"
```

Gatekeeper-Bewertung für einen geeigneten Anwendungstyp prüfen:

```
[RO][FILE][SENS] spctl \
  --assess \
  --type execute \
  --verbose=4 \
  "<AUSFÜHRBARE_DATEI>"
```

Hinweise:

- `codesign` prüft die Signatur, beweist aber nicht die vollständige Dienstfunktion.
- `spctl` ist nicht für jede Art von Binärdatei gleich aussagekräftig.
- Datenschutz- und Sicherheitsmechanismen können den Zugriff eines Daemons einschränken.
- Ein erfolgreicher Start im Terminal beweist nicht die Funktion innerhalb der launchd-Domain.

---

22. macOS-Protokolle eines Startfehlers auswerten

Nach Prozess suchen:

```
[RO][SENS][PRIV] sudo log show \
  --last 1h \
  --predicate 'process == "<PROZESS>"' \
  --style compact \
  --no-pager
```

Nach launchd- und Prozessmeldungen suchen:

```
[RO][SENS][PRIV] sudo log show \
  --last 1h \
  --predicate 'process == "launchd" OR process == "<PROZESS>"' \
  --style compact \
  --no-pager
```

Fehler und Faults anzeigen:

```
[RO][SENS][PRIV] sudo log show \
  --last 1h \
  --predicate '(process == "launchd" OR process == "<PROZESS>") AND (messageType == error OR
messageType == fault)' \
  --style compact \
  --no-pager
```

Live-Beobachtung während eines kontrollierten Startversuchs:

```
[TEST][SENS][PRIV] sudo log stream \
  --timeout 5m \
  --predicate 'process == "launchd" OR process == "<PROZESS>"' \
  --style compact
```

---

23. launchctl-Fehlercode unter macOS übersetzen

Wenn ein `launchctl`-Befehl einen numerischen Fehlercode liefert, kann dieser mit `launchctl error` lesbarer dargestellt werden.

Automatische Einordnung versuchen:

```
[RO] launchctl error <FEHLERCODE>
```

POSIX-Fehlerbereich angeben:

```
[RO] launchctl error posix <FEHLERCODE>
```

Mach-Fehlerbereich angeben:

```
[RO] launchctl error mach <FEHLERCODE>
```

Bootstrap-Fehlerbereich angeben:

```
[RO] launchctl error bootstrap <FEHLERCODE>
```

> Die übersetzte Meldung beschreibt den technischen Fehlerbereich. Die konkrete Ursache muss weiterhin anhand von plist, Programmpfad, Berechtigungen und Protokollen bestimmt werden.

---

24. Kontrollierten Startversuch durchführen

Ein Startversuch verändert den Dienstzustand und kann Folgeprozesse, Netzwerkverbindungen oder automatische Wiederherstellungsaktionen auslösen.

Betriebssystem	Startbefehl
Windows	`[CHANGE][PRIV] Start-Service -Name "<DIENSTNAME>"`
Linux	`[CHANGE][PRIV] sudo systemctl start "<DIENST>"`
macOS systemweit	`[CHANGE][PRIV] sudo launchctl kickstart "system/<LABEL>"`
macOS Benutzerkontext	`[CHANGE] launchctl kickstart "gui/$(id -u)/<LABEL>"`

macOS – PID bei erfolgreichem kickstart ausgeben:

```
[CHANGE][PRIV] sudo launchctl kickstart \
  -p \
  "system/<LABEL>"
```

Ablauf:

```
1. Aktuellen Status und Protokolle sichern
2. Live-Protokollansicht vorbereiten
```

3. Genaue Startzeit notieren
4. Genau einen Startversuch ausführen
5. Rückgabemeldung vollständig erfassen
6. Dienststatus sofort prüfen
7. Prozess-ID und Laufzeit prüfen
8. Listener prüfen
9. Neue Protokollmeldungen sichern
10. Anwendungstest durchführen
11. Keine weiteren Versuche ohne neue Erkenntnis starten

---

25. Manuellen Programmstart nur kontrolliert verwenden

Ein direkter Programmstart kann zusätzliche Fehlermeldungen auf der Konsole sichtbar machen. Er ist aber nicht automatisch sicher oder mit dem Dienststart gleichwertig.

Vorher prüfen:

- erlaubt der Hersteller einen Konsolen- oder Vordergrundmodus?
- ist der Dienstprozess wirklich beendet?
- würde eine zweite Instanz denselben Port oder dieselben Dateien verwenden?
- werden beim Start Datenbanken oder Dateien verändert?
- welches Benutzerkonto muss verwendet werden?
- welche Umgebungsvariablen setzt die Dienstverwaltung?
- welches Arbeitsverzeichnis wird erwartet?
- welche Limits und Sicherheitsrichtlinien gelten?
- existiert ein ausdrücklich dokumentierter Diagnoseparameter?

Der manuelle Start kann sich unterscheiden durch:

Eigenschaft	Dienststart	Manueller Start
Benutzerkonto	Dienstkonto	Aktueller Terminalbenutzer
Arbeitsverzeichnis	Konfiguriert oder systemabhängig	Aktuelles Verzeichnis
Umgebungsvariablen	Dienstspezifisch	Shell- beziehungsweise Benutzervariablen
Berechtigungen	Dienstkontext	Benutzer- oder Administratorkontext
Netzwerkressourcen	Dienstabhängig	Benutzersitzung
Sicherheitsrichtlinien	Dienstmanager und Sandbox	Terminalkontext
Standardausgabe	Journal, Datei oder Ereignisprotokoll	Terminal

> Deshalb wird kein generischer Direktstartbefehl verwendet. Der korrekte Diagnosemodus muss aus der offiziellen Dokumentation der jeweiligen Anwendung stammen.

---

26. Häufige technische Ursachen eines sofortigen Abbruchs

- Syntaxfehler in der Konfiguration,
- fehlende Konfigurationsdatei,
- falscher Programmpfad,
- fehlende Bibliothek oder Laufzeitumgebung,
- falsche Prozessorarchitektur,
- Port bereits belegt,
- ungültiges Zertifikat oder fehlender privater Schlüssel,
- Dienstkonto besitzt keine Leserechte,
- Datenverzeichnis ist nicht beschreibbar,
- Volume oder Netzfreigabe fehlt,
- Datenbank nicht erreichbar,
- DNS-Auflösung fehlerhaft,
- Umgebungsvariable fehlt,
- Secret oder Kennwort ungültig,
- Lizenz fehlt oder ist abgelaufen,
- Datenmigration ist fehlgeschlagen,
- PID-Datei kann nicht erstellt werden,
- Sperrdatei einer alten Instanz ist vorhanden,
- Sicherheitssoftware blockiert die Ausführung,
- Speicher oder Dateideskriptoren sind erschöpft.

---

27. Zeitüberschreitung beim Dienststart analysieren

Ein Timeout bedeutet zunächst, dass ein erwarteter Zustand nicht innerhalb der vorgesehenen Zeit erreicht wurde.

Mögliche Ursachen:

- DNS-Abfrage dauert zu lange,
- Datenbank ist nicht bereit,
- Netzwerkspeicher antwortet nicht,
- Anwendung führt eine lange Migration durch,
- Dienst wartet auf eine gesperrte Datei,

- Portbindung blockiert,
- Dienst meldet seine Bereitschaft nicht korrekt,
- Diensttyp passt nicht zum Programmverhalten,
- Hardware oder Datenträger ist langsam,
- Entschlüsselung oder Zertifikatsprüfung hängt,
- Dienst wartet auf interaktive Eingabe,
- Endlosschleife während der Initialisierung.

Prüfung:

1. Startzeit genau feststellen
2. Letzte Meldung vor dem Timeout ermitteln
3. Prozesszustand während des Wartens prüfen
4. CPU- und I/O-Verhalten beobachten
5. Netzwerkverbindungen des Prozesses prüfen
6. Abhängigkeiten direkt testen
7. Herstellerdokumentation zur Initialisierung prüfen
8. Timeout nicht erhöhen, bevor die Warteursache bekannt ist

> Eine Vergrößerung des Timeouts kann einen langsamen, aber legitimen Start ermöglichen. Sie kann jedoch ebenso einen eigentlichen Fehler nur länger verbergen.

---

28. Portkonflikt als Startursache prüfen

Windows:

```
[RO] Get-NetTCPConnection `
  -State Listen `
  -LocalPort <PORT> `
  -ErrorAction SilentlyContinue |
    Select-Object LocalAddress, LocalPort, OwningProcess
```

Linux:

```
[RO][PRIV] sudo ss -lntp "sport = :<PORT>"
```

macOS:

```
[RO][PRIV] sudo lsof \
  -nP \
  -iTCP:<PORT> \
  -sTCP:LISTEN
```

Wenn der Port belegt ist:

1. PID feststellen.
2. Prozessname bestimmen.
3. Dienstzuordnung prüfen.
4. Bindungsadresse kontrollieren.
5. Prüfen, ob eine alte Instanz läuft.
6. Container- oder Proxybelegung berücksichtigen.
7. Keinen Prozess ungeprüft beenden.
8. Sollkonfiguration und letzte Änderungen prüfen.

---

29. Ressourcenmangel als Startursache prüfen

Aufgabe	Windows	Linux	macOS
Freier Speicherplatz	`[RO] Get-Volume`	`[RO] df -hT`	`[RO] df -h`
Arbeitsspeicher	`[RO] Get-CimInstance Win32_OperatingSystem | Select-Object TotalVisibleMemorySize,FreePhysicalMemory`	`[RO] free -h`	`[RO] vm_stat`
Prozessübersicht	`[RO] Get-Process | Sort-Object WorkingSet64 -Descending`	`[RO] top`	`[RO] top -o mem`
Dateisystemzustand	`[RO] Get-Volume`	`[RO] findmnt`	`[RO] mount`
Offene Dateien des Prozesses	Betriebssystemspezifische Prozesswerkzeuge	`[RO][PRIV] sudo lsof -p <PID>`	`[RO][PRIV] sudo lsof -p <PID>`

Mögliche Startfehler:

- Datenträger vollständig belegt,
- keine freien Inodes unter Linux,
- Arbeitsspeicher erschöpft,

- Auslagerung überlastet,
- Dateisystem schreibgeschützt,
- maximales Prozesslimit erreicht,
- Dateideskriptorlimit erreicht,
- temporäres Verzeichnis nicht verfügbar,
- Laufzeitverzeichnis kann nicht erstellt werden.

---

30. Letzte Änderungen priorisiert untersuchen

Besonders relevant sind Änderungen an:

- Programmversion,
- Dienstdatei oder plist,
- Windows-Dienstkonfiguration,
- systemd-Unit oder Drop-in,
- Dienstkonto oder Kennwort,
- Dateiberechtigungen,
- Zertifikaten,
- Port und Bindungsadresse,
- Firewall,
- Datenbankversion oder Schema,
- Bibliotheken und Laufzeitumgebung,
- Umgebungsvariablen,
- Secrets,
- Mounts und Volumes,
- Containern oder Images,
- Sicherheitssoftware,
- Betriebssystemupdates.

Zeitlicher Vergleich:

```
Letzter erfolgreicher Start
  → Änderung
  → erster fehlgeschlagener Start
```

> Ein zeitlicher Zusammenhang ist ein starker Hinweis, aber noch kein Beweis. Die betroffene Konfiguration oder Komponente muss technisch geprüft werden.

---

31. Typische Fehlinterpretationen

Fehlinterpretation	Richtige Bewertung
„Der Startknopf zeigt einen Fehler, also ist Windows beziehungsweise systemd defekt.“	Die Dienstverwaltung meldet häufig nur den Fehler des gestarteten Programms
„Exitcode 0 bedeutet, der Dienst funktioniert.“	Nur die gemeldete Beendigung wurde als erfolgreich bewertet
„Der Prozess war kurz sichtbar, also startete der Dienst erfolgreich.“	Er kann während der Initialisierung abgebrochen sein
„Manueller Start als Administrator funktioniert, also stimmen die Dienstrechte.“	Der Dienst verwendet einen anderen Kontext
„Mehr Startversuche erhöhen die Chance auf Erfolg.“	Sie können Protokolle überlagern und Startbegrenzungen auslösen
„Timeout erhöhen löst den Fehler.“	Die eigentliche Warteursache bleibt möglicherweise bestehen
„Die Konfigurationsdatei ist syntaktisch gültig, also ist sie fachlich korrekt.“	Pfade, Konten und Backends können weiterhin falsch sein
„Der Port ist frei, also muss der Dienst starten.“	Viele weitere Startvoraussetzungen bleiben ungeprüft
„Ein Neustartzähler beweist einen Softwarefehler.“	Auch fehlende Abhängigkeiten oder Rechte können Wiederholungen auslösen
„Fehlerstatus zurücksetzen repariert den Dienst.“	Nur gespeicherter Zustand und Zähler werden zurückgesetzt

32. Checkliste zur Analyse eines Dienststartfehlers

- [ ] Exakten Dienstnamen beziehungsweise Label bestätigt
- [ ] Richtigen Server und richtige Umgebung bestätigt
- [ ] Aktuellen Status gesichert
- [ ] Exitcode und dienstspezifischen Exitcode erfasst
- [ ] Relevante Protokolle vor neuem Startversuch gesichert
- [ ] Startart und Deaktivierung geprüft
- [ ] systemd-Maskierung beziehungsweise launchd-Override geprüft
- [ ] Programmpfad und Startparameter geprüft
- [ ] Programmdatei vorhanden
- [ ] Dateityp und Architektur geprüft
- [ ] Dienstkonto und Gruppe geprüft
- [ ] Datei- und Verzeichnisrechte geprüft

```
[ ] Arbeitsverzeichnis geprüft
[ ] Umgebungs- und Konfigurationsdateien ermittelt
[ ] Formale Abhängigkeiten geprüft
[ ] Externe Backends geprüft
[ ] Portkonflikt ausgeschlossen
[ ] Speicherplatz und Arbeitsspeicher geprüft
[ ] Neustartregeln und Startbegrenzungen geprüft
[ ] Letzte Änderungen erfasst
[ ] Genau einen kontrollierten Startversuch durchgeführt
[ ] Prozess, PID und Laufzeit danach geprüft
[ ] Listener und Anwendungsfunktion geprüft
[ ] Ursache, Maßnahme und Ergebnis dokumentiert
```

Bewertung des Ergebnisses

Ergebnis	Nächster Schritt
Dienst nicht vorhanden	Installation, Dienstname und Zielsystem prüfen
Dienst deaktiviert oder maskiert	Grund und Sollzustand vor Änderung klären
Programmdatei fehlt	Installation, Update und Dateisystem untersuchen
Zugriff verweigert	Dienstkonto, Rechte und Sicherheitsrichtlinien prüfen
Abhängigkeit schlägt fehl	Betroffene Abhängigkeit separat analysieren
Port ist belegt	Besitzenden Prozess und Konfiguration untersuchen
Konfiguration ist ungültig	Fehlerstelle mit Herstellerwerkzeug bestimmen
Prozess beendet sich mit Exitcode	Produktspezifische Bedeutung des Codes prüfen
Prozess wird durch Signal beendet	Signalursache, Ressourcen und Absturzdiagnose prüfen
Start läuft in Timeout	Letzte Initialisierungsphase und Abhängigkeiten untersuchen
Startgrenze wurde erreicht	Fehlerzustand sichern und Grund der Fehlstarts beheben
Dienst startet und bleibt aktiv	Listener und Anwendungstest durchführen

---

Merksatz

> **Ein Dienststartfehler ist selten mit einem weiteren Startversuch erklärt. Entscheidend ist die erste Phase, in der Sollzustand und**

> **tatsächlicher Startablauf voneinander abweichen.**

Weiterführende Quellen

- Microsoft Learn – Win32_Service
- Microsoft Learn – sc.exe query
- Microsoft Learn – sc.exe qc
- Microsoft Learn – sc.exe qfailure
- Microsoft Learn – Get-WinEvent
- systemd – systemctl
- systemd – systemd.service
- systemd – systemd.exec
- systemd – systemd-analyze
- systemd – journalctl
- Apple – launchctl-Handbuchseite
- Apple – launchd.plist-Handbuchseite
- Apple – plutil-Handbuchseite
- Apple – log-Handbuchseite
