4.8 Konfigurationsfehler untersuchen

Ein Dienst kann installiert, gestartet und über den vorgesehenen Port erreichbar sein, aber aufgrund einer fehlerhaften oder veralteten Konfiguration trotzdem nicht richtig funktionieren.

> **Grundsatz:**
> Nicht irgendeine Konfigurationsdatei prüfen, sondern zuerst feststellen, welche Konfiguration der laufende Dienst tatsächlich verwendet.

Ziele dieser Seite

Nach dieser Seite sollst du:

- Konfigurationsquelle und wirksame Konfiguration unterscheiden können,
- Syntax- und Inhaltsfehler getrennt untersuchen können,
- Konfigurationshierarchien und Überschreibungen erkennen können,
- Dateipfade, Eigentümer und Berechtigungen prüfen können,
- Umgebungsvariablen und Startparameter berücksichtigen können,
- Konfigurationsstände sicher vergleichen können,
- Hashwerte und Änderungszeitpunkte erfassen können,
- versionsabhängige Konfigurationsfehler erkennen können,
- Änderungen kontrolliert vorbereiten und rückgängig machen können,
- Geheimnisse in Konfigurationsdateien schützen können.

---

1. Arten von Konfigurationsfehlern unterscheiden

Fehlerart	Beispiel
Syntaxfehler	Fehlende Klammer oder ungültiges Trennzeichen
Tippfehler	Falscher Schlüsselname oder Hostname
Datentypfehler	Text statt Zahl oder `true` statt erwarteter Zeichenfolge
Wertebereichsfehler	Port außerhalb des gültigen Bereichs
Fehlender Pflichtwert	Datenbankadresse nicht angegeben
Falscher Dateipfad	Zertifikat oder Datenverzeichnis nicht gefunden
Falsche Berechtigung	Dienstkonto kann Datei nicht lesen
Falsche Eigentümerschaft	Datei gehört nicht dem vorgesehenen Konto
Überschreibung	Lokales Drop-in ersetzt den erwarteten Wert

Fehlerart	Beispiel
Versionskonflikt	Option wird von neuer Version nicht mehr unterstützt
Formatfehler	JSON, XML, YAML oder plist ist formal ungültig
Kodierungsfehler	Falsche Zeichenkodierung oder Byte Order Mark
Zeilenendefehler	Windows-Zeilenenden stören ein Unix-Werkzeug
Umgebungsfehler	Variable ist im Terminal, aber nicht im Dienstkontext vorhanden
Geheimnisfehler	Kennwort, Token oder Schlüssel fehlt beziehungsweise ist abgelaufen
Zielsystemfehler	Konfiguration zeigt auf Test- statt Produktivsystem
Reihenfolgefehler	Spätere Datei überschreibt eine frühere Einstellung
Neustart fehlt	Datei wurde geändert, Dienst verwendet aber noch alten Zustand
Reload ungeeignet	Dienst unterstützt kein dynamisches Neuladen
Konfiguration nicht eingebunden	Bearbeitete Datei wird vom Dienst gar nicht geladen

2. Konfigurationsquelle und wirksame Konfiguration unterscheiden

Eine Anwendung kann Einstellungen aus mehreren Quellen beziehen:

Programmstandardwerte
→ Hauptkonfigurationsdatei
→ eingebundene Zusatzdateien
→ lokale Überschreibungen
→ Umgebungsvariablen
→ Startparameter
→ zentrale Konfigurationsverwaltung
→ zur Laufzeit gespeicherte Einstellungen

Später ausgewertete Quellen können frühere Werte überschreiben. Die genaue Priorität ist produktspezifisch.

Begriff	Bedeutung
Standardwert	Vom Programm verwendeter Wert ohne eigene Konfiguration

Begriff	Bedeutung
Quelldatei	Datei, in der eine Einstellung gespeichert ist
Include	Zusätzlich geladene Konfigurationsdatei
Drop-in	Ergänzende oder überschreibende Konfiguration
Startparameter	Beim Programmstart übergebener Wert
Umgebungsvariable	Einstellung aus dem Prozesskontext
Wirksame Konfiguration	Tatsächlich vom Dienst verwendeter Gesamtzustand
Laufzeitkonfiguration	Nach dem Start möglicherweise intern veränderter Zustand

> Eine korrekt aussehende Hauptdatei beweist nicht, dass ihr Wert tatsächlich wirksam ist.

---

3. Vor jeder Prüfung den Konfigurationspfad belegen

Der Konfigurationspfad sollte aus mindestens einer belastbaren Quelle stammen:

1. offizielle Herstellerdokumentation,
2. Dienststartparameter,
3. systemd-Unit beziehungsweise launchd-plist,
4. Windows-Dienstkonfiguration,
5. Anwendungsausgabe zur wirksamen Konfiguration,
6. Installations- oder Deploymentdokumentation,
7. Containerdefinition,
8. Protokollmeldung beim Dienststart.

Nicht ausreichend:

- vermuteter Standardpfad,
- Dateiname einer anderen Installation,
- alte Internetanleitung,
- Konfiguration aus einer anderen Produktversion,
- gleichnamige Datei im Benutzerverzeichnis.

Zu dokumentieren:

```
Dienst:
Produkt und Version:
Konfigurationsquelle:
Tatsächlicher Pfad:
Weitere Includes:
Umgebungsdateien:
Startparameter:
Konfigurationspriorität:
Zeitpunkt der Prüfung:
```

---

4. Konfigurationsquelle unter Windows ermitteln

Dienstpfad und Startparameter anzeigen:

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

Alternative mit sc.exe:

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

Anschließend sind zu prüfen:

- enthält `PathName` einen Konfigurationsparameter?
- wird ein Arbeitsverzeichnis vorausgesetzt?
- verweist der Dienst auf einen Wrapper oder Launcher?
- liest der Wrapper eine weitere Konfiguration?
- wird eine Java-, .NET- oder Skript-Laufzeit aufgerufen?
- stammen Einstellungen aus der Registrierung?
- verwendet die Anwendung ein eigenes Windows-Ereignisprotokoll?
- existieren lokale und globale Konfigurationsdateien?

> Programmpfad und Argumente werden in `PathName` gemeinsam dargestellt. Sie müssen bei der Auswertung sorgfältig getrennt werden.

---

5. Konfigurationsquelle unter Linux mit systemd ermitteln

Wirksame Unit und Drop-ins anzeigen:

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

Wichtige Pfade und Startparameter anzeigen:

```
[RO][SENS] systemctl show "<DIENST>" \
  --property=FragmentPath,DropInPaths,ExecStart,WorkingDirectory,EnvironmentFiles
```

Umgebungswerte anzeigen:

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

Systemweite Konfigurationsdateien eines systemd-Bestandteils zusammenführen:

```
[RO][FILE] systemd-analyze cat-config "<KONFIGURATIONSNAME>"
```

`systemd-analyze cat-config` ist für unterstützte Konfigurationshierarchien von systemd-Komponenten gedacht. Es ist kein universeller Parser für beliebige Anwendungen.

Zu prüfen:

- welche Unit-Datei wurde geladen?
- existieren lokale Drop-ins?
- wird eine Umgebungsdatei verwendet?
- überschreibt `ExecStart` einen Paketstandard?
- wird ein Wrapper oder Startskript verwendet?
- enthält der Dienst einen Konfigurationsparameter?
- wird die Konfiguration über mehrere Dateien zusammengesetzt?

---

6. Konfigurationsquelle unter macOS ermitteln

Systemweiten launchd-Job anzeigen:

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

Benutzerbezogenen Job anzeigen:

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

Bekannte plist lesbar anzeigen:

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

Benutzerbezogene plist anzeigen:

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

Relevante plist-Schlüssel:

```
Program
ProgramArguments
WorkingDirectory
EnvironmentVariables
UserName
GroupName
StandardOutPath
StandardErrorPath
```

Zusätzlich kann eine Anwendung Einstellungen aus dem macOS-Preferences-System verwenden.

Bekannte Preferences-Domain lesend anzeigen:

```
[RO][SENS] defaults read "<DOMAIN>"
```

Einzelnen bekannten Schlüssel lesen:

```
[RO][SENS] defaults read "<DOMAIN>" "<SCHLÜSSEL>"
```

> `defaults read` liest Preferences-Domains. Es ist kein allgemeines Werkzeug für die zuverlässige Bearbeitung beliebiger plist-Dateien.

---

7. Dateiexistenz und Metadaten vergleichen

Aufgabe	Windows	Linux	macOS
Existenz prüfen	`[RO][FILE] Test-Path -LiteralPath "<DATEI>"`	`[RO][FILE] test -f "<DATEI>"`	`[RO][FILE] test -f "<DATEI>"`
Metadaten anzeigen	`[RO][FILE] Get-Item -LiteralPath "<DATEI>"`	`[RO][FILE] stat "<DATEI>"`	`[RO][FILE] stat "<DATEI>"`
Dateityp prüfen	`[RO][FILE] Get-Item -LiteralPath "<DATEI>" | Select-Object FullName,Length,Extension`	`[RO][FILE] file "<DATEI>"`	`[RO][FILE] file "<DATEI>"`
Eigentümer und Rechte	`[RO][FILE][SENS] Get-Acl -LiteralPath "<DATEI>"`	`[RO][FILE] stat -c '%U %G %A %a %n' "<DATEI>"`	`[RO][FILE] stat -f '%Su %Sg %Sp %N' "<DATEI>"`
Symbolischen Link anzeigen	`[RO][FILE] Get-Item -LiteralPath "<DATEI>" | Select-Object FullName,LinkType,Target`	`[RO][FILE] readlink "<DATEI>"`	`[RO][FILE] readlink "<DATEI>"`

Zu prüfen:

- existiert die Datei?
- ist es wirklich eine reguläre Datei?
- handelt es sich um einen symbolischen Link?
- zeigt der Link auf das erwartete Ziel?
- stimmen Eigentümer und Gruppe?
- ist die Datei leer oder ungewöhnlich klein?
- wurde sie zum Störungszeitpunkt verändert?
- liegt sie auf dem erwarteten Dateisystem?
- kann das Dienstkonto alle übergeordneten Verzeichnisse durchlaufen?

> “ Ein Änderungszeitpunkt beweist nur, dass sich Dateimetadaten oder Dateiinhalt geändert haben. Er beweist nicht, wer die fachliche Änderung vorgenommen hat.

---

8. Berechtigungen aller Pfadbestandteile prüfen

Ein Dienst benötigt nicht nur Rechte auf die Datei selbst. Er muss auch die übergeordneten Verzeichnisse erreichen können.

Windows – Datei und übergeordnetes Verzeichnis:

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

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

Linux – vollständigen Pfad zerlegen:

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

Linux – ACL anzeigen:

```
[RO][FILE][SENS] getfacl "<DATEI>"
```

`getfacl` ist nicht auf jeder Minimalinstallation vorhanden.

macOS – Datei und erweiterte ACL anzeigen:

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

macOS – alle Pfadbestandteile prüfen:

```
[RO][FILE] ls -lde "<VERZEICHNIS>" "<DATEI>"
```

Mögliche Fehler:

- Datei ist lesbar, aber Verzeichnis nicht durchsuchbar,
- Dienstkonto darf Konfiguration lesen, aber keine temporäre Datei erzeugen,
- Konfiguration verweist auf ein nicht zugängliches Zertifikat,
- ACL widerspricht den einfachen Dateirechten,
- Eigentümer wurde bei einer Wiederherstellung verändert,
- Datei liegt auf einem schreibgeschützten Volume.

9. Hashwert einer Konfiguration erfassen

Ein kryptografischer Hashwert hilft festzustellen, ob zwei Dateien denselben Inhalt besitzen. Er zeigt nicht, welcher Inhalt fachlich richtig ist.

Betriebssystem	SHA-256-Befehl
Windows	`[RO][FILE] Get-FileHash -Algorithm SHA256 -LiteralPath "<DATEI>"`
Linux	`[RO][FILE] sha256sum "<DATEI>"`
macOS	`[RO][FILE] shasum -a 256 "<DATEI>"`

Mögliche Verwendung:

- Produktiv- und Referenzdatei vergleichen,
- Zustand vor einer Änderung dokumentieren,
- prüfen, ob Deployment wirklich neue Datei ausgeliefert hat,
- Konfigurationsdrift erkennen,
- gesicherte Kopie eindeutig zuordnen.

Wichtig:

- unterschiedliche Hashwerte beweisen einen Inhaltsunterschied,
- gleiche Hashwerte beweisen mit sehr hoher Sicherheit gleichen Dateiinhalt,
- Dateiname, Eigentümer und Berechtigungen werden nicht inhaltlich verglichen,
- Hashwerte geheimer Dateien dürfen je nach Sicherheitsvorgabe ebenfalls geschützt werden.

---

10. Zwei Konfigurationsstände vergleichen

Windows – zeilenweiser Vergleich:

```
[RO][FILE][SENS] Compare-Object `
    (Get-Content -LiteralPath "<REFERENZDATEI>") `
    (Get-Content -LiteralPath "<AKTUELLE_DATEI>")
```

Bedeutung der Seitenindikatoren:

Indikator	Bedeutung
`<=`	Zeile befindet sich nur in der Referenz

Indikator	Bedeutung
`=>`	Zeile befindet sich nur in der aktuellen Datei

Linux:

```
[RO][FILE][SENS] diff -u "<REFERENZDATEI>" "<AKTUELLE_DATEI>"
```

macOS:

```
[RO][FILE][SENS] diff -u "<REFERENZDATEI>" "<AKTUELLE_DATEI>"
```

Hinweise:

- `diff` liefert bei Unterschieden üblicherweise einen Rückgabecode ungleich null, obwohl das Werkzeug technisch korrekt gearbeitet hat.
- Unterschiedliche Reihenfolge kann als Änderung erscheinen.
- Kommentare und Leerzeichen können Unterschiede erzeugen, ohne die Funktion zu verändern.
- Bei strukturierten Formaten sollte zusätzlich die semantische Wirkung geprüft werden.
- Referenzdatei und aktuelle Datei müssen zur gleichen Produktversion passen.
- Secrets dürfen nicht ungefiltert in die Vergleichsausgabe gelangen.

---

11. JSON-Konfiguration prüfen

Windows PowerShell:

```
[RO][FILE][SENS] Test-Json `
  -Json (Get-Content -LiteralPath "<DATEI>.json" -Raw)
```

JSON einlesen und strukturiert anzeigen:

```
[RO][FILE][SENS] Get-Content -LiteralPath "<DATEI>.json" -Raw |
  ConvertFrom-Json
```

Linux mit installiertem jq:

```
[RO][FILE][SENS] jq empty "<DATEI>.json"
```

macOS mit plutil:

```
[RO][FILE] plutil -lint "<DATEI>.json"
```

JSON lesbar ausgeben, ohne Quelldatei zu verändern:

```
[RO][FILE][SENS] plutil \
  -convert json \
  -o - \
  -- "<DATEI>.json"
```

Zu beachten:

- syntaktisch gültiges JSON kann fachlich falsche Werte enthalten,
- doppelte Schlüssel können von Parsern unterschiedlich behandelt werden,
- Kommentare sind im eigentlichen JSON-Standard nicht vorgesehen,
- Datentypen wie Zahl, Zeichenfolge und Boolean müssen stimmen,
- Produktschema und Pflichtfelder müssen zusätzlich geprüft werden.

---

12. XML- und plist-Konfiguration prüfen

Beliebiges XML unter Windows PowerShell einlesen:

```
[RO][FILE][SENS] [xml](
    Get-Content -LiteralPath "<DATEI>.xml" -Raw
)
```

Ein Parserfehler weist auf ungültiges XML hin. Ein erfolgreiches Einlesen beweist noch keine Gültigkeit gegenüber einem Produktschema.

Linux mit installiertem xmllint:

```
[RO][FILE] xmllint --noout "<DATEI>.xml"
```

macOS – plist-Syntax prüfen:

```
[RO][FILE] plutil -lint "<DATEI>.plist"
```

macOS – plist lesbar anzeigen:

```
[RO][FILE][SENS] plutil -p "<DATEI>.plist"
```

macOS – Typ eines bekannten plist-Schlüssels prüfen:

```
[RO][FILE] plutil \
  -type "<SCHLÜSSELPFAD>" \
  "<DATEI>.plist"
```

Zu unterscheiden:

Prüfung	Aussage
XML syntaktisch lesbar	Grundlegende XML-Struktur ist gültig
plist mit `plutil -lint` gültig	Property-List-Format ist syntaktisch gültig
Schema gültig	Felder und Struktur entsprechen einem definierten Schema
Fachlich gültig	Werte funktionieren für das konkrete Produkt
Wirksam	Anwendung verwendet diese Werte tatsächlich

---

13. YAML-, INI- und proprietäre Formate prüfen

Für YAML, INI und herstellerspezifische Formate gibt es kein einzelnes betriebssystemübergreifendes Standardwerkzeug, das gleichzeitig Syntax und fachliche Gültigkeit zuverlässig prüft.

Geeignete Reihenfolge:

1. offizielles Validierungswerkzeug des Produkts verwenden,
2. produktspezifischen Testmodus verwenden,
3. unterstütztes Schema verwenden,
4. Parser nur aus einer vertrauenswürdigen Quelle verwenden,
5. Ausgabe mit der verwendeten Produktversion dokumentieren.

Typische YAML-Fehler:

- falsche Einrückung,
- Tabulator statt Leerzeichen,
- unbeabsichtigte Typumwandlung,
- falsch gesetzte Anführungszeichen,
- ungültige Liste oder Zuordnung,
- doppelte Schlüssel,
- nicht aufgelöste Referenz,
- mehrere Dokumente in einer Datei.

Typische INI-Fehler:

- falscher Abschnitt,
- doppelter Schlüssel,
- unerwartete Groß- und Kleinschreibung,
- falsches Trennzeichen,
- ungültige Escape-Sequenz,
- Wert im falschen Abschnitt,
- Kommentarzeichen wird als Wert interpretiert.

> Keine beliebigen Online-Validatoren für produktive Konfigurationen verwenden. Dateien können interne Adressen, Konten, Tokens oder andere Geheimnisse enthalten.

---

14. systemd-Unit-Dateien und Überschreibungen prüfen

Wirksame Unit anzeigen:

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

Unit formal prüfen:

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

Lokale Abweichungen von ausgelieferten Unit-Dateien anzeigen:

```
[RO][FILE][SENS] systemd-delta
```

Nur Abweichungen für systemd-Systemkonfigurationen anzeigen:

```
[RO][FILE][SENS] systemd-delta --type=overridden,extended
```

Mögliche Kategorien von `systemd-delta`:

Kategorie	Bedeutung
`overridden`	Ursprüngliche Datei wurde vollständig überschrieben
`extended`	Drop-in ergänzt oder verändert Einstellungen
`masked`	Unit ist maskiert
`equivalent`	Dateien sind inhaltlich gleichwertig
`redirected`	Datei verweist auf ein anderes Ziel

Besonders prüfen:

- wurde `ExecStart` in einem Drop-in ersetzt?
- blieb eine alte Überschreibung nach einem Update bestehen?
- verweist `EnvironmentFile` auf eine alte Datei?
- wurde eine Hersteller-Unit unter `/usr` direkt verändert?
- enthält ein Drop-in eine leere Zuweisung, die einen früheren Wert zurücksetzt?
- wurde eine Unit versehentlich maskiert?

---

15. launchd-plist unter macOS prüfen

Syntax prüfen:

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

Struktur lesbar anzeigen:

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

Label gezielt auslesen:

```
[RO][FILE] plutil \
  -extract Label raw \
  "/Library/LaunchDaemons/<LABEL>.plist"
```

Programmpfad gezielt auslesen:

```
[RO][FILE][SENS] plutil \
  -extract Program raw \
  "/Library/LaunchDaemons/<LABEL>.plist"
```

Falls kein `Program`-Schlüssel vorhanden ist, kann der Programmpfad im ersten Element von `ProgramArguments` stehen.

Erstes Argument auslesen:

```
[RO][FILE][SENS] plutil \
  -extract ProgramArguments.0 raw \
  "/Library/LaunchDaemons/<LABEL>.plist"
```

Zu prüfen:

- Dateiname und `Label` stimmen sinnvoll überein,
- `Program` oder erstes `ProgramArguments`-Element ist korrekt,
- Programmpfad ist absolut,
- Arbeitsverzeichnis existiert,
- Dienstkonto und Gruppe existieren,
- Ausgabe- und Fehlerpfade sind beschreibbar,
- Datentypen der Schlüssel stimmen,
- Startbedingungen widersprechen sich nicht,
- Datei liegt in der richtigen launchd-Domain.

---

16. Umgebungsvariablen als Fehlerquelle untersuchen

Eine Variable kann in der interaktiven Shell vorhanden sein, aber im Dienstkontext fehlen.

Typische verwendete Variablen:

```
PATH
HOME
TEMP
TMP
LANG
LC_ALL
HTTP_PROXY
HTTPS_PROXY
NO_PROXY
DATABASE_URL
CONFIG_PATH
CERT_PATH
```

Windows – Umgebungsvariablen der aktuellen Sitzung:

```
[RO][SENS] Get-ChildItem Env:
```

Windows – maschinenweite Variable lesen:

```
[RO][SENS] [Environment]::GetEnvironmentVariable(
    "<VARIABLE>",
    "Machine"
)
```

Linux – für systemd konfigurierte Dienstvariablen anzeigen:

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

macOS – launchd-Job und plist prüfen:

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

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

Wichtig:

- `sudo` kann Umgebungsvariablen verändern oder entfernen,
- Systemdienste besitzen häufig ein reduziertes `PATH`,
- `$HOME` kann fehlen oder auf ein anderes Verzeichnis zeigen,
- Proxyvariablen können nur in einer Benutzersitzung gesetzt sein,
- Variablennamen können unter Unix-Systemen Groß- und Kleinschreibung unterscheiden,
- Variablen können Secrets enthalten.

> Die vollständige Umgebung eines produktiven Prozesses darf nicht ungeprüft ausgegeben oder weitergegeben werden.

---

17. Relative Pfade und Arbeitsverzeichnis prüfen

Folgende Konfiguration kann problematisch sein:

```
config/application.conf
certificates/server.crt
logs/application.log
./data
```

Relative Pfade werden vom Arbeitsverzeichnis des Prozesses aus aufgelöst. Dieses kann beim Dienststart anders sein als beim manuellen Start.

Windows – Dienstpfad anzeigen:

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

Linux – Arbeitsverzeichnis anzeigen:

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

macOS – plist prüfen:

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

Prüffragen:

- enthält die Konfiguration relative Pfade?
- welches Arbeitsverzeichnis verwendet der Dienst?
- existiert der Pfad in diesem Kontext?
- besitzt das Dienstkonto Zugriff?
- zeigt ein symbolischer Link auf das erwartete Ziel?
- verhält sich der manuelle Start nur deshalb anders?

> Für Serverdienste sind eindeutig dokumentierte absolute Pfade meist leichter nachvollziehbar. Ob sie unterstützt werden, entscheidet jedoch die Produktkonfiguration.

---

18. Zeichenkodierung und Zeilenenden prüfen

Windows – Anfangsbytes einer Datei anzeigen:

```
[RO][FILE] Format-Hex -LiteralPath "<DATEI>" |
    Select-Object -First 5
```

Linux – Dateityp und mögliche Kodierungshinweise:

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

macOS:

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

Typische Probleme:

- UTF-8 mit oder ohne Byte Order Mark,
- UTF-16 statt UTF-8,
- ungültige Bytefolge,
- Windows-Zeilenende `CRLF`,
- Unix-Zeilenende `LF`,
- unsichtbares Steuerzeichen,
- nicht brechendes Leerzeichen,

- Tabulator in YAML,
- typografische statt gerader Anführungszeichen,
- falsche Normalisierung von Unicode-Zeichen.

Unsichtbare Zeichen unter Linux oder macOS darstellen:

```
[RO][FILE][SENS] sed -n 'l' "<DATEI>"
```

> Eine Konvertierung verändert die Datei und darf erst nach Sicherung, Freigabe und Kenntnis des erwarteten Formats durchgeführt werden.

---

19. Include-Dateien und Konfigurationshierarchie prüfen

Eine Hauptdatei kann weitere Dateien laden:

```
Hauptdatei
  → conf.d/*.conf
  → lokale Überschreibung
  → geheime Umgebungsdatei
  → automatisch generierte Datei
```

Zu prüfen:

- welche Include-Anweisungen existieren?
- in welcher Reihenfolge werden Dateien geladen?
- werden Platzhalter oder Wildcards verwendet?
- existiert eine alte Sicherungsdatei mit passender Endung im Include-Verzeichnis?
- wird eine automatisch generierte Datei manuell überschrieben?
- gibt es environment-, mandanten- oder hostabhängige Dateien?
- wird eine Datei aufgrund falscher Endung ignoriert?
- verweist ein Include auf einen nicht vorhandenen Pfad?
- wird dieselbe Einstellung mehrfach definiert?
- gewinnt der erste oder der letzte Wert?

>

> Die Lade- und Überschreibungsreihenfolge muss aus der Dokumentation des konkreten Produkts entnommen werden.

---

20. Versionskompatibilität der Konfiguration prüfen

Nach einem Update können Einstellungen:

- umbenannt,
- entfernt,
- als veraltet markiert,
- in einen anderen Abschnitt verschoben,
- mit verändertem Datentyp erwartet,
- mit neuem Standardwert versehen,
- sicherheitsbedingt deaktiviert,
- durch ein neues Format ersetzt worden sein.

Zu dokumentieren:

```
Vorherige Produktversion:
Aktuelle Produktversion:
Version der Konfigurationsvorlage:
Datum der letzten Konfigurationsänderung:
Migrationshinweise gelesen:
Veraltete Optionen:
Neue Pflichtwerte:
Geänderte Standardwerte:
```

Prüfquellen:

1. offizielle Versionshinweise,
2. Upgrade- oder Migrationsanleitung,
3. Beispielkonfiguration derselben Version,
4. produktspezifischer Konfigurationstest,
5. Startprotokoll mit Warnungen zu veralteten Optionen.

> Eine Beispieldatei aus einer neueren oder älteren Version darf nicht ungeprüft als Referenz verwendet werden.

21. Secrets und Zugangsdaten sicher behandeln

Konfigurationsdateien können enthalten:

- Kennwörter,
- API-Schlüssel,
- Datenbank-Verbindungszeichenfolgen,
- private Schlüssel,
- Zugriffstoken,
- Sitzungsschlüssel,
- LDAP-Bindekennwörter,
- Cloud-Zugangsdaten,
- interne URLs mit eingebetteten Zugangsdaten.

Regeln:

```
[ ] Keine vollständige Geheimdatei in ein Ticket kopieren
[ ] Keine Secrets in Terminalbefehle mit Protokollierung einsetzen
[ ] Keine produktive Konfiguration in Online-Validatoren hochladen
[ ] Keine privaten Schlüssel mit Vergleichsausgaben veröffentlichen
[ ] Nur bereinigte Kopien weitergeben
[ ] Originaldatei geschützt aufbewahren
[ ] Dateirechte vor und nach einer Maßnahme prüfen
[ ] Kompromittierte Geheimnisse nicht nur zurückändern, sondern rotieren
```

Unsichere Darstellung:

```
DATABASE_URL=postgres://admin:Kennwort@db-server/database
```

Bereinigte Darstellung:

```
DATABASE_URL=postgres://<BENUTZER>:<ENTFERNT>@<DB-SERVER>/<DATENBANK>
```

22. Containerkonfiguration berücksichtigen

Bei Containern können Einstellungen stammen aus:

- Image-Standardwerten,
- Umgebungsvariablen,
- Compose- oder Orchestrator-Definition,
- Secrets,
- Config-Objekten,
- eingebundenen Dateien,
- persistenten Volumes,
- Startbefehl oder Entrypoint,
- Reverse-Proxy-Konfiguration.

Prüffragen:

- welche Image-Version wird tatsächlich ausgeführt?
- wurde der Container nach einer Änderung neu erstellt?
- ist die richtige Datei in den Container eingebunden?
- wurde eine Datei oder ein ganzes Verzeichnis gemountet?
- verdeckt ein Mount eine im Image vorhandene Datei?
- verwendet der Container die erwarteten Umgebungsvariablen?
- enthält der Startbefehl eine Überschreibung?
- wurde ein Secret aktualisiert?
- ist die Konfiguration persistent?
- unterscheidet sich die Hostdatei von der Datei im Container?
- verwendet die Anwendung intern einen anderen Pfad?

> Eine Änderung an einer Hostdatei ist nur wirksam, wenn genau diese Datei in den Container eingebunden und von der Anwendung verwendet wird.

---

23. Konfiguration vor einer Änderung beweissicher erfassen

Mindestens erfassen:

Information	Zweck
Vollständiger Pfad	Eindeutige Dateiidentifikation
Dateigröße	Plausibilitätsprüfung
Änderungszeit	Zeitlicher Zusammenhang
Eigentümer und Rechte	Zugriffskontrolle
SHA-256-Hash	Inhaltsidentifikation
Produktversion	Kompatibilitätsprüfung

Information	Zweck
Dienststartparameter	Nachweis der Verwendung
Include-Dateien	Vollständige Konfigurationskette
Wirksame Werte	Tatsächlicher Zustand
Geheimnisstatus	Schutzbedarf
Sicherungspfad	Rückfallmöglichkeit

Dokumentationsvorlage:

```
Dienst:
Produktversion:
Konfigurationsdatei:
Weitere Konfigurationsquellen:
Eigentümer:
Berechtigungen:
Änderungszeit:
SHA-256:
Wirksame Konfiguration geprüft:
Syntaxprüfung:
Herstellervalidierung:
Auffälliger Wert:
Letzte bekannte funktionierende Version:
Geplante Änderung:
Rückfallmöglichkeit:
```

---

24. Kontrollierter Änderungsablauf

Eine Konfigurationsänderung besteht nicht nur aus dem Bearbeiten einer Datei.

```
1. Ursache und Sollwert bestimmen
2. Produktversion und Dokumentation prüfen
3. Originalzustand dokumentieren
4. Geschützte Sicherung erstellen
5. Genau eine fachliche Änderung durchführen
6. Syntax validieren
```

7. Produktspezifischen Konfigurationstest ausführen
8. Unterschied zur Ausgangsversion prüfen
9. Eigentümer und Berechtigungen kontrollieren
10. Reload oder Neustart nach Dokumentation planen
11. Protokolle während der Aktivierung beobachten
12. Technischen und fachlichen Funktionstest durchführen
13. Nebenwirkungen prüfen
14. Bei Fehlschlag kontrolliert zurückrollen
15. Ursache, Änderung und Ergebnis dokumentieren

Risiken:

- eine Sicherung im aktiven Include-Verzeichnis wird mitgeladen,
- Editor verändert Eigentümer oder Berechtigungen,
- temporäre Datei erhält falsche Zugriffsrechte,
- Reload wird unterstützt, übernimmt aber nicht alle Werte,
- Neustart unterbricht aktive Transaktionen,
- alte Konfiguration passt nicht mehr zur aktualisierten Software,
- Rückrollen der Datei reicht nach einer Datenmigration nicht aus.

---

25. Reload, Neustart und Neuerstellung unterscheiden

Maßnahme	Wirkung
Reload	Dienst liest unterstützte Konfigurationsbereiche neu ein
Neustart	Prozess wird beendet und neu gestartet
Systemneustart	Gesamtes Betriebssystem startet neu
Containerneustart	Gleiche Containerinstanz wird erneut gestartet
Containerneuerstellung	Container wird aus Definition und Image neu erzeugt
Redeployment	Anwendung und Konfiguration werden erneut bereitgestellt

Prüffragen:

- unterstützt der Dienst einen Reload?
- welche Einstellungen werden beim Reload übernommen?
- benötigt die Änderung einen vollständigen Neustart?
- wird die Datei nur beim Installieren eingelesen?
- wird Konfiguration in eine Datenbank importiert?

- wird der Container bei einer Neuerstellung mit identischen Parametern erzeugt?
- bleiben Volumes und Secrets erhalten?
- gehen Diagnoseinformationen beim Neustart verloren?

> Nicht jeder Dienst übernimmt jede Einstellung durch einen Reload. Die unterstützte Aktivierungsmethode muss aus der Produktdokumentation stammen.

---

26. Typische Fehlinterpretationen

Fehlinterpretation	Richtige Bewertung
„Die Datei ist syntaktisch gültig, also funktioniert die Konfiguration.“	Werte, Pfade und Abhängigkeiten können trotzdem falsch sein
„Ich habe die Standarddatei geprüft.“	Der Dienst kann eine andere Datei verwenden
„Die Datei wurde geändert, also nutzt der Prozess den neuen Wert.“	Reload oder Neustart kann fehlen
„Der Wert steht nur einmal in der Hauptdatei.“	Include, Variable oder Startparameter kann ihn überschreiben
„Manueller Start funktioniert.“	Die Shell kann andere Variablen und Rechte besitzen
„Gleicher Dateiname bedeutet gleiche Konfiguration.“	Inhalt muss mit Hash oder Vergleich geprüft werden
„Der neueste Änderungszeitpunkt zeigt den Verursacher.“	Er zeigt nur eine Dateiänderung
„Die Beispielkonfiguration ist immer korrekt.“	Sie muss zur eingesetzten Version passen
„Eine Sicherungsdatei im Konfigurationsordner ist ungefährlich.“	Wildcard-Includes können sie mitladen
„Zurückkopieren der Datei stellt alles wieder her.“	Daten-, Schema- oder Laufzeitänderungen können bestehen bleiben

---

27. Checkliste zur Konfigurationsanalyse

[ ] Richtigen Dienst und richtige Produktversion bestätigt

[ ] Tatsächliche Konfigurationsquelle ermittelt

[ ] Startparameter geprüft

```
[ ] Umgebungsvariablen berücksichtigt
[ ] Hauptdatei und Includes erfasst
[ ] Überschreibungsreihenfolge geklärt
[ ] Wirksame Konfiguration ermittelt
[ ] Datei vorhanden und nicht leer
[ ] Symbolische Links geprüft
[ ] Eigentümer und Berechtigungen geprüft
[ ] Arbeitsverzeichnis berücksichtigt
[ ] Syntax mit geeignetem Werkzeug geprüft
[ ] Produktspezifische Validierung durchgeführt
[ ] Pflichtwerte und Datentypen geprüft
[ ] Pfade und Zielsysteme geprüft
[ ] Port- und Protokollwerte geprüft
[ ] Versionskompatibilität geprüft
[ ] Referenzdatei gehört zur gleichen Version
[ ] Hash und Änderungszeit dokumentiert
[ ] Unterschiede zur funktionierenden Version geprüft
[ ] Secrets geschützt
[ ] Ausgangszustand gesichert
[ ] Genau eine Änderung geplant
[ ] Aktivierungsmethode geklärt
[ ] Rückfallmöglichkeit vorbereitet
[ ] Funktionstest festgelegt
```

Bewertung des Ergebnisses

Ergebnis	Nächster Schritt
Falsche Datei wurde geprüft	Tatsächliche Konfigurationsquelle untersuchen
Syntax ungültig	Fehlerstelle sichern und nach Produktschema korrigieren
Syntax gültig, Wert aber falsch	Sollwert und Abhängigkeit prüfen
Datei wird überschrieben	Konfigurationspriorität und übergeordnete Quelle prüfen
Dienst verwendet alte Werte	Unterstützten Reload oder Neustart planen
Pfad existiert nicht	Deployment, Mount und Dateisystem prüfen
Dienstkonto kann Datei nicht lesen	Berechtigung und Eigentümer untersuchen
Option ist veraltet	Migrationsdokumentation der eingesetzten Version prüfen
Umgebungsvariable fehlt	Tatsächlichen Dienstkontext prüfen

Ergebnis	Nächster Schritt
Konfigurationen unterscheiden sich	Änderung fachlich bewerten und Ursache bestimmen
Konfiguration ist korrekt und wirksam	Berechtigungen, Ressourcen und Anwendung untersuchen

---

Merksatz

> **Nicht die Datei, die ein Administrator bearbeitet, ist entscheidend, sondern die Konfiguration, die der Dienst im tatsächlichen Laufzeitkontext wirksam verwendet.**

Weiterführende Quellen

- Microsoft Learn – Get-CimInstance
- Microsoft Learn – Get-FileHash
- Microsoft Learn – Compare-Object
- Microsoft Learn – Test-Json
- Microsoft Learn – Get-Acl
- systemd – systemctl
- systemd – systemd-analyze
- systemd – systemd-delta
- systemd – systemd.unit
- Apple – plutil-Handbuchseite
- Apple – defaults-Handbuchseite
- Apple – launchctl-Handbuchseite
- Apple – launchd.plist-Handbuchseite
