6.1 Service Configuration Management – Ziele, Begriffe und Grundlagen

> **Kurz erklärt**
>
> Service Configuration Management sorgt dafür, dass wichtige Informationen über Services, Systeme, Komponenten und deren Beziehungen nachvollziehbar gepflegt werden.
>
> Ziel ist, zu wissen:
>
> - welche Komponenten vorhanden sind,
> - wie sie zusammenhängen,
> - wer verantwortlich ist,
> - welche Versionen genutzt werden,
> - und welche Services von welchen Bestandteilen abhängig sind.
>
> Dadurch können Incidents, Problems, Changes und Releases besser bewertet und gesteuert werden.

Warum Service Configuration Management wichtig ist

IT-Services bestehen selten aus nur einer einzelnen Komponente.

Ein scheinbar einfacher Service kann abhängig sein von:

- Anwendung,
- Datenbank,
- Server,
- Netzwerk,
- DNS,
- Zertifikaten,
- Identitätsdienst,
- Storage,
- Cloud-Ressourcen,
- Schnittstellen,
- Lieferanten,
- Monitoring,
- Backup,
- und Berechtigungen.

Wenn diese Zusammenhänge unbekannt sind, entstehen Risiken:

- Incidents werden langsamer analysiert,
- Changes werden falsch bewertet,
- Auswirkungen werden unterschätzt,
- Verantwortlichkeiten bleiben unklar,
- Problems werden nicht richtig eingegrenzt,
- Dokumentation ist veraltet,
- und Serviceabhängigkeiten werden erst im Fehlerfall sichtbar.

Service Configuration Management schafft dafür eine nachvollziehbare Informationsbasis.

Was ist eine Configuration?

Eine Configuration beschreibt, wie ein Service oder System aufgebaut ist.

Dazu gehören beispielsweise:

- eingesetzte Komponenten,
- Versionen,
- Einstellungen,
- Abhängigkeiten,
- Beziehungen,
- Standorte,
- Verantwortlichkeiten,
- Lieferanten,
- technische Schnittstellen,
- und unterstützende Services.

Beispiel:

Der Service „VPN-Zugang“ besteht nicht nur aus dem VPN-Gateway.

Er kann abhängig sein von:

- VPN-Client,
- Zertifikaten,
- Identity Provider,
- MFA,
- Firewall,
- DNS,
- Netzwerkverbindung,
- Monitoring,
- und Benutzergruppen.

Configuration Item

Ein **Configuration Item (CI)** ist ein Bestandteil, der für die Bereitstellung oder Verwaltung eines Service relevant ist.

Beispiele:

- Server,
- Anwendung,
- Datenbank,
- Netzwerkgerät,
- Firewall-Regel,
- Zertifikat,
- Cloud-Ressource,
- Benutzergruppe,
- Schnittstelle,
- Lizenz,
- Dokumentation,
- Lieferantenvertrag,
- Arbeitsplatzgerät,
- Drucker,
- Service selbst.

Nicht jedes technische Objekt muss automatisch ein CI sein.

Die Organisation muss festlegen, welche Elemente wichtig genug sind, um verwaltet zu werden.

Beispiele für Configuration Items

CI-Typ	Beispiel
Service	Mitarbeiterportal
Anwendung	Warenwirtschaft
Server	APP-WEB-01
Datenbank	SQL-Cluster Produktion
Netzwerk	Standort-Firewall
Cloud	Azure App Service
Sicherheit	TLS-Zertifikat
Identität	Entra-ID-Gruppe
Dokumentation	Betriebsrunbook
Lieferant	Supportvertrag Provider
Arbeitsplatz	Notebook
Peripherie	Etikettendrucker

Welche CI-Typen sinnvoll sind, hängt von Organisation, Services und Steuerungsbedarf ab.

Ziel von Service Configuration Management

Service Configuration Management soll sicherstellen, dass verlässliche Informationen verfügbar sind über:

- Services,
- Configuration Items,
- Beziehungen,
- Versionen,
- Status,
- Eigentümer,
- Standorte,
- Lieferanten,
- Abhängigkeiten,
- und Änderungen.

Diese Informationen unterstützen andere ITIL Practices.

Beispiele:

- Incident Management erkennt betroffene Services schneller.
- Problem Management erkennt gemeinsame Ursachen.
- Change Enablement bewertet Auswirkungen besser.
- Release Management kennt betroffene Komponenten.
- Service Level Management versteht Serviceabhängigkeiten.
- Information Security Management erkennt kritische Assets.

Service Configuration Management ist mehr als Inventarisierung

Inventarisierung beantwortet oft nur:

> Was ist vorhanden?

Service Configuration Management beantwortet zusätzlich:

> Wie hängt es zusammen und warum ist es für den Service wichtig?

Beispiel:

Eine Inventarliste zeigt:

- Server APP-01,
- Datenbank DB-01,
- Firewall FW-01.

Service Configuration Management zeigt zusätzlich:

- APP-01 betreibt das Mitarbeiterportal.
- APP-01 nutzt DB-01.
- FW-01 ermöglicht Zugriff vom Standort Süd.
- Das Mitarbeiterportal ist kritisch für interne Anträge.
- Service Owner ist die interne IT.
- Provider X unterstützt die Firewall.

Diese Beziehungen machen die Informationen im Betrieb wertvoll.

CI-Beziehungen

Beziehungen zwischen CIs sind besonders wichtig.

Typische Beziehungen:

- Service nutzt Anwendung,
- Anwendung nutzt Datenbank,
- Datenbank nutzt Storage,
- Server läuft auf Virtualisierungsplattform,
- Anwendung ist abhängig von Identity Provider,
- Standort nutzt Firewall,
- Zertifikat schützt Webservice,
- Monitoring überwacht Server,
- Lieferant unterstützt Komponente,
- Benutzergruppe erhält Zugriff auf Anwendung.

Beziehungen helfen zu verstehen, welche Auswirkungen ein Ausfall oder Change haben kann.

Beispiel: Serviceabhängigkeiten

```
Benutzer
   ↓
Mitarbeiterportal
   ↓
Webserver
```

```
   ↓
Anwendungsdienst
   ↓
Datenbank
   ↓
Storage
```

Zusätzlich abhängig von:

- DNS,
- Zertifikat,
- Identity Provider,
- Netzwerk,
- Monitoring,
- Backup,
- Firewall.

Wenn die Datenbank ausfällt, ist das Mitarbeiterportal betroffen.

Wenn das Zertifikat abläuft, kann die Anmeldung scheitern.

Wenn DNS fehlerhaft ist, erscheint der Service möglicherweise komplett unerreichbar.

CMDB

Eine **Configuration Management Database (CMDB)** ist ein System oder Datenbestand, in dem Configuration Items und ihre Beziehungen verwaltet werden.

Eine CMDB kann enthalten:

- CI-Name,
- CI-Typ,
- Status,
- Version,
- Umgebung,
- Standort,
- Owner,
- Lieferant,
- Kritikalität,
- Beziehungen,
- letzte Änderung,
- verknüpfte Incidents,
- verknüpfte Problems,
- verknüpfte Changes,

- Dokumentationslinks.

Eine CMDB muss nicht zwingend ein einzelnes großes Werkzeug sein.

Wichtig ist, dass die benötigten Informationen zuverlässig verfügbar und gepflegt sind.

CMDB und CMS unterscheiden

Der Begriff **Configuration Management System (CMS)** kann breiter verstanden werden als eine einzelne CMDB.

Ein CMS kann mehrere Datenquellen verbinden.

Beispiele:

- CMDB,
- Asset Management,
- Monitoring,
- Cloud-Inventar,
- Netzwerkdokumentation,
- Virtualisierungsplattform,
- Endpoint Management,
- Softwareverteilung,
- Dokumentationssystem,
- ITSM-Tool.

Die CMDB kann ein Teil dieses größeren Systems sein.

Configuration Management und Asset Management unterscheiden

Configuration Management	Asset Management
Fokus auf Servicebeziehungen und Betriebsrelevanz	Fokus auf Besitz, Kosten, Lebenszyklus und Vertragsdaten
betrachtet CIs und Abhängigkeiten	betrachtet Assets als wirtschaftliche oder verwaltete Objekte
wichtig für Incident, Problem und Change	wichtig für Einkauf, Lizenzierung und Lifecycle
Beispiel: Datenbank ist abhängig von Storage	Beispiel: Server wurde am 12.03.2025 gekauft

Ein Objekt kann gleichzeitig Asset und CI sein.

Beispiel:

Ein produktiver Server ist wirtschaftlich ein Asset und betrieblich ein CI.

Welche Informationen sind sinnvoll?

Nicht jedes Detail muss erfasst werden.

Sinnvolle CI-Informationen können sein:

- eindeutiger Name,
- CI-Typ,
- Beschreibung,
- Servicezuordnung,
- Umgebung,
- Status,
- Version,
- Standort,
- Owner,
- Supportgruppe,
- Lieferant,
- Kritikalität,
- Beziehungen,
- Dokumentationslink,
- letzte Prüfung,
- letzte Änderung,
- Sicherheitsklassifizierung.

Zu viele ungenutzte Felder machen Pflege schwer.

Zu wenige Informationen machen die Daten im Betrieb wertlos.

CI-Status

Ein CI kann unterschiedliche Status besitzen.

Beispiele:

- geplant,
- bestellt,
- im Aufbau,
- aktiv,
- in Wartung,
- außer Betrieb,
- ersetzt,
- archiviert.

Der Status hilft zu erkennen, ob ein CI produktiv genutzt wird oder nicht.

Beispiel:

Ein alter Server darf nicht mehr als aktive Abhängigkeit eines produktiven Services erscheinen, wenn er bereits außer Betrieb genommen wurde.

Umgebungen kennzeichnen

CIs sollten ihrer Umgebung zugeordnet werden.

Beispiele:

- Entwicklung,
- Test,
- Abnahme,
- Staging,
- Produktion,
- Disaster Recovery,
- Labor,
- Schulung.

Das ist wichtig, weil ein Change an einem Testsystem anders bewertet wird als ein Change an einem produktiven System.

Unklare Umgebungskennzeichnung kann zu gefährlichen Verwechslungen führen.

Kritikalität erfassen

Die Kritikalität beschreibt, wie wichtig ein CI oder Service für die Organisation ist.

Beispiele:

- geschäftskritisch,
- hoch,
- mittel,
- niedrig,
- unterstützend,
- nicht produktiv.

Die Kritikalität hilft bei:

- Priorisierung von Incidents,
- Risikobewertung von Changes,
- Notfallplanung,
- Monitoring,
- Patchplanung,
- Lieferantensteuerung,
- und Service Reviews.

Wichtig:

Die Kritikalität eines CIs sollte aus der Servicebedeutung abgeleitet werden.

Owner und Verantwortlichkeiten

Für wichtige CIs sollte klar sein:

- Wer ist fachlich verantwortlich?
- Wer ist technisch verantwortlich?
- Wer betreibt das CI?
- Wer darf Änderungen genehmigen?
- Wer pflegt die Dokumentation?
- Wer ist Ansprechpartner bei Incidents?
- Welcher Lieferant unterstützt das CI?

Ohne Owner veralten Informationen schnell.

Ungeeignet:

> Niemand fühlt sich für das System verantwortlich.

Besser:

> Service Owner, technischer Owner und Supportgruppe sind dokumentiert.

Service Owner und CI Owner unterscheiden

Rolle	Bedeutung
Service Owner	Verantwortung für den gesamten Service und dessen Wertbeitrag
CI Owner	Verantwortung für ein bestimmtes Configuration Item
Supportgruppe	bearbeitet Incidents, Changes oder Betriebsaufgaben
Lieferant	unterstützt oder betreibt bestimmte Komponenten

Ein Service kann viele CIs besitzen.

Ein CI kann mehrere Services unterstützen.

Deshalb sind Beziehungen und Verantwortlichkeiten wichtig.

Datenqualität

Service Configuration Management ist nur hilfreich, wenn die Datenqualität ausreichend ist.

Typische Probleme:

- veraltete Einträge,
- fehlende Beziehungen,
- falsche Owner,
- doppelte CIs,
- unklare Namen,
- fehlende Versionen,
- nicht dokumentierte Changes,
- außer Betrieb genommene Systeme bleiben aktiv,
- Cloud-Ressourcen werden nicht erfasst,
- manuelle Pflege wird vergessen.

Schlechte Datenqualität kann gefährlicher sein als keine Daten, weil sie falsche Sicherheit erzeugt.

Datenqualität prüfen

Mögliche Prüffragen:

- Existiert das CI noch?
- Ist der Status korrekt?
- Ist der Owner aktuell?
- Stimmen Version und Umgebung?
- Sind Beziehungen vollständig?
- Ist der betroffene Service verknüpft?
- Wurde der letzte Change berücksichtigt?
- Sind Lieferant und Supportgruppe korrekt?
- Gibt es Dubletten?
- Wird das CI noch aktiv genutzt?

Regelmäßige Prüfung ist notwendig, besonders bei kritischen Services.

Discovery

Discovery bezeichnet das automatische Erkennen von Systemen, Komponenten oder technischen Eigenschaften.

Beispiele:

- Server erkennen,
- installierte Software erfassen,
- Netzwerkgeräte finden,
- Cloud-Ressourcen inventarisieren,
- offene Ports erkennen,
- Zertifikate finden,
- Versionen auslesen,
- Beziehungen ableiten.

Discovery kann helfen, Daten aktuell zu halten.

Es ersetzt aber nicht jede fachliche Bewertung.

Ein Tool kann erkennen, dass ein Server existiert.

Es weiß nicht automatisch, welcher Geschäftsprozess davon abhängig ist.

Manuelle Pflege

Manche Informationen müssen manuell oder organisatorisch gepflegt werden.

Beispiele:

- Service Owner,
- fachliche Kritikalität,
- Geschäftsprozessbezug,
- Lieferantenverantwortung,
- genehmigte Ausnahme,
- Risikoakzeptanz,
- Dokumentationslink,
- geplante Außerbetriebnahme.

Automatische Erkennung und manuelle Pflege müssen zusammenarbeiten.

Namenskonventionen

Klare Namenskonventionen helfen, CIs eindeutig zu identifizieren.

Beispiele:

- Standort,
- Umgebung,
- Funktion,
- laufende Nummer,
- Servicebezug.

Ungeeignet:

- Server1,
- Testneu,
- Alt-System,
- Backup-neu-final,
- App2.

Besser:

- PRD-APP-PORTAL-01,
- TST-DB-ERP-02,
- BER-FW-STANDORT-01.

Die konkrete Namenskonvention muss zur Organisation passen.

Configuration Baseline

Eine Configuration Baseline beschreibt einen bekannten, freigegebenen Zustand.

Beispiele:

- Standardkonfiguration eines Servers,
- genehmigte Version einer Anwendung,
- geprüfte Firewall-Regelbasis,
- freigegebene Client-Konfiguration,
- definierter Stand eines Release-Pakets.

Baselines helfen bei:

- Vergleich mit aktuellem Zustand,
- Audit,
- Fehlersuche,
- Rollback,
- Compliance,
- Standardisierung.

Wenn ein System von der Baseline abweicht, sollte bekannt sein, warum.

Abweichungen erkennen

Abweichungen können entstehen durch:

- manuelle Änderungen,
- ungeplante Changes,
- Notfallmaßnahmen,

- fehlerhafte Automatisierung,
- Lieferantenänderungen,
- Schatten-IT,
- Konfigurationsdrift,
- fehlende Dokumentation.

Abweichungen sind nicht automatisch falsch.

Sie müssen aber nachvollziehbar und bewertet sein.

Service Configuration Management und Incident Management

Bei Incidents helfen Configuration-Daten zu verstehen:

- welcher Service betroffen ist,
- welche Komponenten beteiligt sind,
- welche Abhängigkeiten bestehen,
- wer zuständig ist,
- welche Changes zuletzt durchgeführt wurden,
- welche Known Errors existieren,
- ob weitere Services betroffen sein könnten.

Beispiel:

Eine Anwendung ist nicht erreichbar.

Durch CI-Beziehungen wird sichtbar, dass Anwendung, Datenbank und Identity Provider relevant sind.

Die Diagnose wird schneller und gezielter.

Service Configuration Management und Problem Management

Problem Management nutzt Configuration-Daten, um Ursachen zu erkennen.

Beispiele:

- mehrere Incidents betreffen dieselbe Version,
- mehrere Services hängen vom gleichen Datenbankcluster ab,
- ein Standort hat wiederkehrende Netzwerkprobleme,
- mehrere Anwendungen nutzen dasselbe Zertifikat,
- gleiche Hardwaremodellserie verursacht Ausfälle.

Ohne Beziehungen bleiben solche Muster oft verborgen.

Service Configuration Management und Change Enablement

Bei Changes helfen Configuration-Daten bei der Auswirkungsbewertung.

Zu prüfen ist:

- Welche CIs werden geändert?
- Welche Services hängen davon ab?
- Welche Benutzergruppen sind betroffen?
- Welche anderen Changes laufen parallel?
- Welche Owner müssen beteiligt werden?
- Welche Dokumentation muss aktualisiert werden?
- Welche Risiken entstehen?

Ein Change ohne Kenntnis der Abhängigkeiten kann unerwartete Incidents erzeugen.

Service Configuration Management und Release Management

Release und Deployment benötigen verlässliche Informationen über:

- Zielumgebungen,
- Versionen,
- Komponenten,
- Abhängigkeiten,
- unterstützte Plattformen,
- Konfigurationen,
- Schnittstellen,
- und Rollback-Zustände.

Nach einem Release müssen CI-Daten aktualisiert werden.

Sonst stimmen spätere Analysen nicht mehr.

Service Configuration Management und Information Security Management

Configuration-Daten unterstützen Sicherheitsarbeit.

Beispiele:

- kritische Systeme identifizieren,
- veraltete Versionen erkennen,
- fehlende Patches priorisieren,
- Zertifikate überwachen,
- externe Schnittstellen erfassen,
- Administratorzugänge nachvollziehen,

- Datenklassifizierung berücksichtigen,
- Shadow IT erkennen,
- Notfallpläne erstellen.

Sicherheitsmaßnahmen hängen stark davon ab, zu wissen, welche Systeme und Abhängigkeiten existieren.

Praxisbeispiel: Abgelaufenes Zertifikat

Incident

Benutzer können sich nicht am Mitarbeiterportal anmelden.

Analyse

Das TLS-Zertifikat ist abgelaufen.

Configuration Management hilft durch:

- Verknüpfung des Zertifikats mit dem Mitarbeiterportal,
- Owner des Zertifikats,
- Ablaufdatum,
- betroffene Services,
- Abhängigkeit zum Identity Provider,
- Link zum Erneuerungs-Runbook.

Verbesserung

Zertifikate werden künftig als CIs mit Ablaufdatum, Owner und Monitoring erfasst.

Praxisbeispiel: Datenbankcluster

Situation

Mehrere Anwendungen melden Fehler.

Ohne CI-Beziehungen

Jedes Anwendungsteam sucht getrennt.

Mit CI-Beziehungen

Es wird sichtbar:

- alle betroffenen Anwendungen nutzen denselben Datenbankcluster,
- der Datenbankcluster nutzt denselben Storage,

- Storage zeigt Fehler.

Nutzen

Die Ursache wird schneller eingegrenzt.

Praxisbeispiel: Change-Auswirkung

Change

Firewall-Regel soll angepasst werden.

Configuration Management zeigt:

- welche Services über diese Regel kommunizieren,
- welche Standorte betroffen sind,
- welche Schnittstellen genutzt werden,
- welcher Service Owner beteiligt werden muss,
- welche Monitoringprüfungen nach dem Change notwendig sind.

Nutzen

Der Change wird besser bewertet und sicherer umgesetzt.

Typische Fehler

Fehler 1

CMDB wird als reine Inventarliste verstanden.

Fehler 2

CIs werden erfasst, aber Beziehungen fehlen.

Fehler 3

Daten werden einmal erstellt und danach nicht gepflegt.

Fehler 4

Owner sind unbekannt oder veraltet.

Fehler 5

Changes aktualisieren die CMDB nicht.

Fehler 6

Discovery-Daten werden ungeprüft übernommen.

Fehler 7

Zu viele unwichtige Details werden gepflegt, aber wichtige Servicebeziehungen fehlen.

Fehler 8

Cloud-Ressourcen und externe Services werden nicht berücksichtigt.

Fehler 9

CMDB-Daten werden in Incident, Problem und Change nicht genutzt.

Fehler 10

Dubletten und uneinheitliche Namen erschweren die Nutzung.

Fehler 11

Kritikalität wird technisch statt geschäftlich bewertet.

Fehler 12

Dokumentation und CMDB widersprechen sich.

Checkliste CI erfassen

- [ ] eindeutiger Name vergeben
- [ ] CI-Typ festgelegt
- [ ] betroffener Service verknüpft
- [ ] Umgebung dokumentiert
- [ ] Status gesetzt
- [ ] Version oder Konfiguration erfasst
- [ ] Standort oder Plattform dokumentiert

- [ ] Owner benannt
- [ ] Supportgruppe benannt
- [ ] Lieferant erfasst, falls relevant
- [ ] Kritikalität bewertet
- [ ] Beziehungen zu anderen CIs erfasst
- [ ] Dokumentationslink ergänzt

Checkliste Beziehungen prüfen

- [ ] Service zu Anwendung verknüpft
- [ ] Anwendung zu Datenbank verknüpft
- [ ] Anwendung zu Identity Provider verknüpft
- [ ] Anwendung zu Netzwerk oder Firewall verknüpft
- [ ] Zertifikate verknüpft
- [ ] Schnittstellen verknüpft
- [ ] Monitoring verknüpft
- [ ] Backup oder Wiederherstellung berücksichtigt
- [ ] Lieferantenbezug verknüpft
- [ ] wichtige Benutzergruppen oder Standorte berücksichtigt

Checkliste Datenqualität

- [ ] CI existiert tatsächlich
- [ ] Status ist aktuell
- [ ] Owner ist aktuell
- [ ] Version stimmt
- [ ] Umgebung stimmt
- [ ] Beziehungen sind plausibel
- [ ] keine Dublette vorhanden
- [ ] letzter Change wurde berücksichtigt
- [ ] Dokumentationslink funktioniert
- [ ] Kritikalität ist nachvollziehbar
- [ ] nicht mehr genutzte CIs sind archiviert

Bedeutung für Fachinformatiker für Systemintegration

Fachinformatiker arbeiten täglich mit Configuration Items.

Beispiele:

- Server,
- Netzwerke,
- Firewalls,
- Clients,
- Drucker,
- Benutzergruppen,
- Zertifikate,
- Storage,
- Backups,
- Monitoring,
- Cloud-Ressourcen.

Wichtig ist nicht nur, diese Komponenten technisch zu betreiben.

Wichtig ist auch zu dokumentieren:

- wozu sie gehören,
- welche Services davon abhängen,
- wer verantwortlich ist,
- welche Version genutzt wird,
- welche Änderungen erfolgt sind,
- und welche Risiken bestehen.

Gute Configuration-Daten machen Fehleranalyse, Changes und Servicebetrieb deutlich sicherer.

Zusammenfassung

> Service und Komponenten identifizieren
> ↓
> wichtige Configuration Items festlegen
> ↓
> Eigenschaften und Owner dokumentieren
> ↓
> Beziehungen zwischen CIs erfassen
> ↓
> Datenqualität prüfen
> ↓
> Changes und Releases einbeziehen
> ↓
> Informationen in Incident, Problem und Change nutzen

↓
CMDB oder CMS kontinuierlich pflegen und verbessern

Merksätze

> Service Configuration Management ist mehr als Inventarisierung.

> Beziehungen zwischen CIs sind oft wichtiger als einzelne technische Details.

> Eine CMDB ist nur so wertvoll wie ihre Datenqualität.

> Discovery hilft, ersetzt aber keine fachliche Verantwortung.

> Ohne aktuelle Configuration-Daten werden Incidents, Problems und Changes riskanter.

> Jede wichtige Änderung sollte auch die Configuration-Daten aktualisieren.

Verwandte Seiten

- 6.2 Configuration Items und Beziehungen
- 6.3 Configuration Management Database
- 6.4 Discovery, Pflege und Datenqualität
- 6.5 Service Configuration Management im Zusammenspiel mit Incident-, Problem- und Change-Management
- Incident Management
- Problem Management
- Change Enablement
- Release Management
- Information Security Management

Quellen und Versionsstand

Offizielle Grundlagen

- PeopleCert – ITIL Practice Guide: Service Configuration Management
- PeopleCert – ITIL Practice Guide: IT Asset Management
- PeopleCert – ITIL Practice Guide: Change Enablement
- PeopleCert – ITIL Practice Guide: Incident Management
- PeopleCert – ITIL Practice Guide: Problem Management
- ITIL Foundation – Version 5

Einordnung

Die dargestellten:

- CI-Beispiele,
- CMDB-Inhalte,
- Beziehungsmodelle,
- Checklisten,
- Praxisbeispiele,
- und Datenqualitätskriterien

sind herstellerneutrale Praxisempfehlungen.

ITIL schreibt keine universelle:

- CMDB-Struktur,
- CI-Typenliste,
- Pflichtfeldliste,
- Namenskonvention,
- Discovery-Pflicht,
- oder konkrete Toolauswahl

für alle Organisationen vor.

Die konkrete Umsetzung muss an:

- Services,
- Risiken,
- Organisationsgröße,
- Toollandschaft,
- Datenqualität,
- Sicherheitsanforderungen,
- Supportmodell,
- Change-Modell,
- und verfügbare Fähigkeiten

angepasst werden.

Behandelter Framework-Stand: ITIL Version 5
Zusätzlich berücksichtigt: aktuelle ITIL-4-Practice-Guidance
Fachlicher Stand: August 2026
