# 6.3 Configuration Management Database

> **Kurz erklärt**
>
> Eine **Configuration Management Database (CMDB)** ist ein Datenbestand oder Werkzeug, in dem Configuration Items und ihre Beziehungen verwaltet werden.
>
> Sie hilft dabei, Services, Systeme, Komponenten, Versionen, Verantwortlichkeiten und Abhängigkeiten nachvollziehbar zu dokumentieren.
>
> Der Wert einer CMDB entsteht nicht durch möglichst viele Einträge, sondern durch aktuelle, verlässliche und nutzbare Informationen für Incident, Problem, Change, Release und Security Management.

---

**Warum eine CMDB wichtig ist**

IT-Services bestehen aus vielen abhängigen Bestandteilen.

Ohne zentrale oder verlässliche Configuration-Daten entstehen typische Probleme:

- niemand weiß genau, welcher Service betroffen ist,
- Changes werden mit unvollständiger Auswirkungsbewertung durchgeführt,
- Incidents werden langsamer analysiert,
- Problems werden nicht richtig eingegrenzt,
- veraltete Systeme bleiben unentdeckt,
- Zertifikate laufen ab,
- Abhängigkeiten zu Lieferanten fehlen,
- Dokumentation widerspricht der Realität,
- und Verantwortlichkeiten bleiben unklar.

Eine CMDB soll diese Informationen strukturiert nutzbar machen.

---

**Was eine CMDB leisten soll**

Eine CMDB soll helfen zu beantworten:

- Welche Services gibt es?
- Welche Configuration Items gehören dazu?
- Welche Systeme sind produktiv?
- Welche Versionen sind im Einsatz?
- Welche Komponenten hängen voneinander ab?

- Welche Changes wurden zuletzt durchgeführt?
- Welche Incidents betreffen ein CI?
- Welche Problems oder Known Errors sind verknüpft?
- Wer ist verantwortlich?
- Welche Lieferanten sind beteiligt?
- Welche Dokumentation gehört dazu?
- Welche Risiken bestehen?

Damit unterstützt die CMDB nicht nur Dokumentation, sondern tägliche Betriebsentscheidungen.

---

**CMDB ist kein Selbstzweck**

Eine CMDB ist nur sinnvoll, wenn sie genutzt wird.

Ungeeignet:

> Wir erfassen alles, weil eine CMDB vollständig sein muss.

Besser:

> Wir erfassen die Informationen, die für Servicebetrieb, Support, Changes, Risiken und Entscheidungen wirklich benötigt werden.

Eine zu große, ungepflegte CMDB wird schnell unübersichtlich und unzuverlässig.

Eine kleine, gut gepflegte CMDB kann deutlich nützlicher sein.

---

**CMDB und Configuration Management System**

Eine CMDB ist ein Datenbestand für Configuration Items und Beziehungen.

Ein **Configuration Management System (CMS)** kann breiter sein.

Ein CMS kann mehrere Quellen verbinden, zum Beispiel:

- CMDB,
- Asset Management,
- Monitoring,
- Cloud-Inventar,
- Endpoint Management,
- Netzwerkdokumentation,

- Softwareverteilung,
- Virtualisierungsplattform,
- Dokumentationssystem,
- ITSM-Tool,
- Security-Werkzeuge.

Die CMDB kann also Teil eines größeren Configuration Management Systems sein.

---

### CMDB und Inventarliste unterscheiden

| Inventarliste | CMDB |
|---|---|
| zeigt häufig nur vorhandene Geräte oder Assets | zeigt Configuration Items und Servicebeziehungen |
| Fokus auf Bestand | Fokus auf Betrieb, Abhängigkeiten und Wirkung |
| Beispiel: Server existiert | Beispiel: Server betreibt Service X und nutzt Datenbank Y |
| wichtig für Übersicht | wichtig für Incident, Problem, Change und Risiko |
| oft objektbezogen | service- und beziehungsbezogen |

Eine Inventarliste beantwortet:

> Was haben wir?

Eine CMDB beantwortet zusätzlich:

> Wofür wird es genutzt und wovon hängt es ab?

---

### Typische Inhalte einer CMDB

Eine CMDB kann enthalten:

- Configuration Items,
- CI-Typen,
- CI-Status,
- Servicezuordnung,
- Versionen,
- Umgebungen,
- Standorte,
- Owner,
- Supportgruppen,

- Lieferanten,
- Kritikalität,
- Sicherheitsklassifizierung,
- Beziehungen,
- Dokumentationslinks,
- verknüpfte Incidents,
- verknüpfte Problems,
- verknüpfte Changes,
- verknüpfte Releases,
- letzte Prüfung,
- letzte Änderung.

Nicht jede CMDB benötigt alle Felder.

Die Felder müssen zum Zweck und Pflegeaufwand passen.

---

### Beispiel für einen CMDB-Eintrag

| Feld | Beispiel |
|---|---|
| **CI-Name** | PRD-APP-PORTAL-01 |
| **CI-Typ** | Server |
| **Status** | aktiv |
| **Umgebung** | Produktion |
| **Service** | Mitarbeiterportal |
| **Owner** | Plattform-Team |
| **Supportgruppe** | Application Support |
| **Version** | Linux Server Version X |
| **Standort** | Rechenzentrum Berlin |
| **Kritikalität** | hoch |
| **Lieferant** | interner Betrieb |
| **Dokumentation** | Betriebsrunbook Mitarbeiterportal |
| **Beziehungen** | läuft Portal-App, nutzt Datenbank PRD-DB-PORTAL-01 |
| **letzte Prüfung** | 02.08.2026 |

Dieses Beispiel ist bewusst einfach gehalten.

Die tatsächlichen Felder hängen von Organisation und Werkzeug ab.

---

### Wichtige CI-Typen in einer CMDB

Mögliche CI-Typen:

- Business Service,
- Technical Service,
- Anwendung,
- Server,
- Datenbank,
- Netzwerkgerät,
- Firewall,
- Cloud-Ressource,
- Zertifikat,
- DNS-Eintrag,
- Benutzergruppe,
- Schnittstelle,
- Monitoringregel,
- Backupjob,
- Dokumentation,
- Lieferantenvertrag,
- Arbeitsplatzgerät,
- Drucker.

Wichtig ist eine klare interne Definition.

Wenn Teams CI-Typen unterschiedlich verwenden, werden Auswertungen ungenau.

---

**Service als zentrales Element**

Eine CMDB sollte nicht nur technische Komponenten sammeln.

Wichtig ist die Verbindung zum Service.

Beispiel:

```
Mitarbeiterportal
   ↓ nutzt
Portal-Anwendung
   ↓ läuft auf
PRD-APP-PORTAL-01
   ↓ nutzt
PRD-DB-PORTAL-01
   ↓ nutzt
PRD-STO-01
```

Zusätzlich:

- Zertifikat,
- DNS,
- Identity Provider,
- Monitoring,
- Backup,
- Supportgruppe,
- Service Owner.

Erst diese Serviceperspektive macht die CMDB im Betrieb wertvoll.

---

**Beziehungen in der CMDB**

Beziehungen sind einer der wichtigsten Bestandteile einer CMDB.

Typische Beziehungen:

- Service nutzt Anwendung,
- Anwendung läuft auf Server,
- Anwendung nutzt Datenbank,
- Server nutzt Storage,
- Service nutzt Zertifikat,
- Anwendung kommuniziert mit Schnittstelle,
- Standort nutzt Firewall,
- Monitoring überwacht CI,
- Backupjob sichert Datenbank,
- Lieferant unterstützt Komponente.

Ohne Beziehungen ist eine CMDB oft nur eine Inventarliste.

---

**Beispiel: Beziehungskette**

```
Benutzer
   ↓
Mitarbeiterportal
   ↓
Webserver
   ↓
Anwendung
   ↓
Datenbank
   ↓
```

Storage

Weitere Abhängigkeiten:

- DNS,
- Zertifikat,
- Identity Provider,
- MFA,
- Firewall,
- Monitoring,
- Backup,
- Lieferant.

Wenn eine Komponente ausfällt, hilft die CMDB zu erkennen, welche Services betroffen sein können.

---

**CMDB und Incident Management**

Bei Incidents hilft eine CMDB durch:

- schnelle Erkennung betroffener Services,
- Zuordnung zu verantwortlichen Teams,
- Sicht auf Abhängigkeiten,
- Anzeige letzter Changes,
- Verknüpfung zu Known Errors,
- Priorisierung nach Kritikalität,
- bessere Eskalation,
- schnellere Kommunikation.

Beispiel:

Ein Benutzer meldet, dass das Mitarbeiterportal nicht erreichbar ist.

Die CMDB zeigt:

- Portal läuft auf PRD-APP-PORTAL-01,
- nutzt PRD-DB-PORTAL-01,
- ist abhängig vom Identity Provider,
- letzter Change betraf das Zertifikat.

Dadurch kann die Diagnose gezielter beginnen.

---

**CMDB und Problem Management**

Problem Management nutzt CMDB-Daten, um gemeinsame Ursachen zu erkennen.

Beispiele:

- mehrere Incidents betreffen dieselbe Version,
- mehrere Services hängen vom gleichen Datenbankcluster ab,
- viele Probleme treten an einem Standort auf,
- mehrere Anwendungen nutzen dasselbe Zertifikat,
- wiederkehrende Störungen betreffen denselben Lieferanten.

Ohne CMDB-Beziehungen werden solche Muster oft erst spät sichtbar.

---

**CMDB und Change Enablement**

Change Enablement benötigt CMDB-Daten für die Auswirkungsbewertung.

Zu prüfen ist:

- Welche CIs werden geändert?
- Welche Services hängen davon ab?
- Welche Benutzer oder Standorte sind betroffen?
- Welche Owner müssen einbezogen werden?
- Welche anderen Changes betreffen dieselben CIs?
- Welche Dokumentation muss aktualisiert werden?
- Welche Tests sind notwendig?
- Welche Risiken entstehen?

Ein Change ohne CMDB-Prüfung kann unerwartete Auswirkungen verursachen.

---

**CMDB und Release Management**

Release und Deployment benötigen verlässliche Informationen über:

- Zielumgebungen,
- betroffene Komponenten,
- Versionen,
- Abhängigkeiten,
- Schnittstellen,
- Konfigurationen,
- Rollback-Zustände,
- Monitoring,
- Dokumentation.

Nach einem Release müssen CMDB-Daten aktualisiert werden.

Sonst zeigt die CMDB einen veralteten Zustand.

---

**CMDB und Information Security Management**

Eine CMDB unterstützt Sicherheitsarbeit durch Informationen über:

- kritische Systeme,
- externe Erreichbarkeit,
- veraltete Versionen,
- Zertifikate,
- Datenklassifizierung,
- privilegierte Systeme,
- Schnittstellen,
- Lieferanten,
- Cloud-Ressourcen,
- betroffene Services bei Schwachstellen.

Beispiel:

Eine Sicherheitslücke betrifft eine bestimmte Softwareversion.

Mit CMDB-Daten kann schneller erkannt werden, welche Systeme und Services betroffen sind.

---

**CMDB und Service Level Management**

Service Level Management kann CMDB-Daten nutzen, um zu verstehen:

- welche Komponenten kritische Services unterstützen,
- welche Abhängigkeiten Service Levels gefährden,
- welche Lieferanten beteiligt sind,
- welche CIs besonders hohe Verfügbarkeit benötigen,
- welche Changes Service Levels beeinflussen können.

Ein Service Level ist schwer zu steuern, wenn die technischen Abhängigkeiten unbekannt sind.

---

**CMDB-Datenquellen**

Mögliche Datenquellen für eine CMDB:

- manuelle Pflege,
- Discovery-Werkzeuge,
- Monitoring,
- Asset Management,
- Endpoint Management,
- Cloud-Plattformen,
- Virtualisierungsplattformen,
- Netzwerkmanagement,

- Softwareverteilung,
- ITSM-Tickets,
- Change Records,
- Lieferanteninformationen,
- Dokumentationssysteme.

Keine einzelne Datenquelle ist automatisch vollständig.

Die CMDB muss aus passenden Quellen aufgebaut und geprüft werden.

---

**Automatische Discovery**

Discovery kann technische Informationen automatisch erfassen.

Beispiele:

- Server,
- IP-Adressen,
- installierte Software,
- Betriebssystemversionen,
- Cloud-Ressourcen,
- Zertifikate,
- offene Dienste,
- Netzwerkgeräte,
- virtuelle Maschinen,
- Container,
- Datenbankinstanzen.

Discovery ist hilfreich für Aktualität.

Aber Discovery erkennt nicht immer:

- fachliche Kritikalität,
- Service Owner,
- Geschäftsprozessbezug,
- Risikoakzeptanz,
- Dokumentationsqualität,
- geplante Ablösung,
- oder organisatorische Verantwortung.

---

**Manuelle Pflege**

Manuelle Pflege bleibt wichtig für Informationen wie:

- Service Owner,

- fachliche Kritikalität,
- Supportmodell,
- Lieferantenverantwortung,
- Sicherheitsklassifizierung,
- Geschäftsprozessbezug,
- Runbook-Link,
- Notfallinformationen,
- Risikoakzeptanz,
- geplante Außerbetriebnahme.

Diese Informationen benötigen Verantwortliche.

Ohne klare Pflegeverantwortung veralten sie schnell.

---

**Datenqualität in der CMDB**

Eine CMDB ist nur so gut wie ihre Datenqualität.

Wichtige Qualitätskriterien:

- Vollständigkeit,
- Aktualität,
- Eindeutigkeit,
- Richtigkeit,
- Konsistenz,
- Nachvollziehbarkeit,
- Nutzbarkeit,
- Verantwortlichkeit.

Schlechte Daten können gefährlich sein.

Sie führen zu falschen Entscheidungen, weil Teams der CMDB vertrauen, obwohl sie veraltet ist.

---

**Typische Datenqualitätsprobleme**

| Problem | Folge |
|---|---|
| doppelte CIs | Incidents und Changes werden falsch verknüpft |
| veraltete Owner | Eskalation dauert länger |
| fehlende Beziehungen | Auswirkung wird unterschätzt |
| falscher Status | außer Betrieb genommene Systeme erscheinen aktiv |
| fehlende Versionen | Sicherheitsbewertung wird ungenau |
| fehlende Servicezuordnung | Priorisierung wird schwieriger |

| Problem | Folge |
| --- | --- |
| falsche Umgebung | Test und Produktion werden verwechselt |
| fehlende Dokumentationslinks | Betrieb dauert länger |

---

### Dubletten vermeiden

Dubletten entstehen, wenn dasselbe CI mehrfach erfasst wird.

Beispiele:

- APP-01,
- app01,
- Portal Server,
- PRD-APP-PORTAL-01.

Folgen:

- Beziehungen verteilen sich auf mehrere Einträge,
- Reports werden falsch,
- Changes erscheinen unvollständig,
- Incidents werden nicht richtig gebündelt,
- Owner widersprechen sich.

Gegenmaßnahmen:

- Namenskonvention,
- eindeutige IDs,
- Dublettenprüfung,
- klare Erfassungsregeln,
- regelmäßige Datenbereinigung.

---

### Namenskonventionen

Eine gute Namenskonvention unterstützt Eindeutigkeit.

Mögliche Bestandteile:

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

Beispiele:

- PRD-APP-PORTAL-01,
- TST-DB-ERP-02,
- BER-FW-STANDORT-01,
- PRD-CERT-PORTAL-2026.

Die Namenskonvention sollte einfach genug sein, damit sie konsequent genutzt wird.

---

**Pflichtfelder und optionale Felder**

Nicht jedes Feld sollte verpflichtend sein.

Zu viele Pflichtfelder führen dazu, dass Benutzer falsche Werte eintragen, nur um den Datensatz speichern zu können.

Sinnvolle Pflichtfelder können sein:

- CI-Name,
- CI-Typ,
- Status,
- Umgebung,
- Owner oder Supportgruppe,
- Servicezuordnung bei produktiven CIs.

Optionale Felder können abhängig vom CI-Typ sein.

Beispiel:

Ein Zertifikat benötigt Ablaufdatum.

Ein Drucker benötigt Standort.

Eine Anwendung benötigt Version und Service Owner.

---

**CMDB-Modell**

Das CMDB-Modell beschreibt, welche CI-Typen, Attribute und Beziehungen verwendet werden.

Zu klären ist:

- Welche CI-Typen gibt es?
- Welche Attribute sind notwendig?
- Welche Beziehungen sind erlaubt?
- Welche Felder sind Pflicht?
- Welche Datenquellen werden genutzt?
- Wer darf CIs erstellen?
- Wer darf CIs ändern?

- Wie werden Dubletten verhindert?
- Wie wird Qualität geprüft?

Ein einfaches, verstandenes Modell ist besser als ein komplexes Modell, das niemand pflegt.

---

**Granularität**

Granularität beschreibt, wie detailliert CIs erfasst werden.

Beispiel:

Sehr grob:

- Service: Mitarbeiterportal

Mittel:

- Service,
- Anwendung,
- Webserver,
- Datenbank,
- Zertifikat.

Sehr fein:

- einzelne Konfigurationsdateien,
- einzelne Firewall-Regeln,
- einzelne Bibliotheken,
- einzelne Schnittstellenparameter.

Die richtige Granularität hängt vom Nutzen ab.

Zu grob hilft bei technischer Analyse wenig.

Zu fein ist schwer pflegbar.

---

**Beispiel für passende Granularität**

Für einen kritischen Webservice sinnvoll:

- Service,
- Anwendung,
- Datenbank,
- Server,
- Load Balancer,
- Zertifikat,

- DNS,
- Identity Provider,
- Monitoring,
- Backup,
- wichtige Schnittstellen.

Nicht immer sinnvoll:

- jede einzelne Konfigurationszeile,
- jede temporäre Datei,
- jedes unkritische Testobjekt.

Die CMDB soll Entscheidungen unterstützen, nicht jede technische Kleinigkeit abbilden.

---

**CMDB-Lebenszyklus**

Ein CI durchläuft einen Lebenszyklus.

```
geplant
   ↓
erstellt oder beschafft
   ↓
in Betrieb genommen
   ↓
geändert
   ↓
regelmäßig geprüft
   ↓
außer Betrieb genommen
   ↓
archiviert oder gelöscht
```

Die CMDB sollte diesen Lebenszyklus abbilden.

Besonders wichtig ist die saubere Außerbetriebnahme.

---

**Außerbetriebnahme von CIs**

Vor dem Entfernen eines CIs sollte geprüft werden:

- Welche Services nutzen das CI?
- Gibt es noch aktive Beziehungen?

- Gibt es offene Incidents oder Problems?
- Gibt es geplante Changes?
- Wird das CI noch überwacht?
- Gibt es Backups oder Archivdaten?
- Gibt es Lizenz- oder Vertragsbezug?
- Muss Dokumentation angepasst werden?
- Muss ein Lieferant informiert werden?

Ein CI einfach zu löschen kann spätere Nachvollziehbarkeit zerstören.

Oft ist Archivierung sinnvoller.

---

**CMDB und Change-Prozess**

Changes sollten CMDB-Daten aktualisieren.

Beispiele:

- neue Version nach Update,
- neue Beziehung nach Schnittstellenänderung,
- neues Zertifikat nach Erneuerung,
- neuer Owner nach Übergabe,
- neuer Status nach Außerbetriebnahme,
- neue Cloud-Ressource nach Deployment,
- neue Monitoringbeziehung nach Alarmregel.

Wenn Changes die CMDB nicht aktualisieren, veraltet sie automatisch.

---

**CMDB und Dokumentation**

Die CMDB ersetzt nicht jede Dokumentation.

Sie kann aber auf Dokumentation verweisen.

Beispiele:

- Betriebsrunbook,
- Architekturdiagramm,
- Notfallhandbuch,
- Installationsanleitung,
- Schnittstellendokumentation,
- Lieferantenvertrag,
- Sicherheitskonzept,
- Backup- und Restore-Anleitung.

Die CMDB zeigt, welche Dokumentation zu welchem CI oder Service gehört.

---

**CMDB und Monitoring**

Monitoring und CMDB sollten sich ergänzen.

Monitoring zeigt:

- Zustand,
- Alarme,
- Verfügbarkeit,
- Performance,
- Kapazität,
- Fehlerereignisse.

CMDB zeigt:

- Servicezuordnung,
- Kritikalität,
- Owner,
- Beziehungen,
- Version,
- Dokumentation.

Beispiel:

Monitoring meldet Ausfall eines Servers.

Die CMDB zeigt, welche Services davon abhängen und wer informiert werden muss.

---

**CMDB und Cloud**

Cloud-Umgebungen verändern sich oft schnell.

Herausforderungen:

- Ressourcen entstehen automatisch,
- Ressourcen werden schnell gelöscht,
- Namen sind uneinheitlich,
- Tags fehlen,
- Owner sind unklar,
- Kostenstellen fehlen,
- Beziehungen sind dynamisch,
- externe Dienste sind beteiligt.

Für Cloud-Ressourcen sind klare Tags, automatisierte Erkennung und regelmäßige Prüfung besonders wichtig.

---

### CMDB und Tags

Tags oder Labels können helfen, Cloud- und Plattformressourcen zuzuordnen.

Mögliche Tags:

- Service,
- Umgebung,
- Owner,
- Kostenstelle,
- Kritikalität,
- Datenklasse,
- Ablaufdatum,
- Projekt,
- Supportgruppe.

Beispiel:

| Tag | Wert |
|---|---|
| Service | Mitarbeiterportal |
| Umgebung | Produktion |
| Owner | Plattform-Team |
| Kritikalität | hoch |
| Datenklasse | intern |

Tags ersetzen keine vollständige CMDB, können aber wichtige Daten liefern.

---

### CMDB und Container

Auch Container-Umgebungen können CMDB-relevant sein.

Mögliche CIs:

- Container-Plattform,
- Cluster,
- Namespace,
- Anwendung,
- Image-Version,
- Datenbank,
- Volume,

- Ingress,
- Secret,
- Zertifikat,
- Monitoring,
- Backupjob.

Wichtig ist, nicht jeden kurzlebigen Container einzeln dauerhaft zu pflegen.

Oft sind Service, Anwendung, Plattform, Version und Abhängigkeiten wichtiger als einzelne temporäre Instanzen.

---

**CMDB und Schnittstellen**

Schnittstellen werden oft vergessen.

Dabei sind sie für Services kritisch.

Zu erfassen sind:

- Quellsystem,
- Zielsystem,
- Protokoll,
- Authentifizierung,
- Zertifikat,
- Datenformat,
- Owner,
- Lieferant,
- Kritikalität,
- Monitoring,
- bekannte Fehler,
- Abhängigkeiten.

Ein Schnittstellenfehler kann mehrere Services gleichzeitig betreffen.

---

**CMDB und Zertifikate**

Zertifikate sollten bei wichtigen Services als relevante CIs betrachtet werden.

Sinnvolle Informationen:

- Name,
- betroffener Service,
- Hostname,
- Ablaufdatum,
- Aussteller,

- Owner,
- Erneuerungsprozess,
- Monitoring,
- abhängige Systeme,
- Dokumentationslink.

Viele vermeidbare Incidents entstehen durch abgelaufene oder falsch erneuerte Zertifikate.

---

**Berechtigungen und Gruppen als CIs**

Auch Benutzergruppen oder Berechtigungskonzepte können CI-relevant sein.

Beispiele:

- Administratorgruppe,
- Rollen für Fachanwendung,
- VPN-Zugriffsgruppe,
- MFA-Ausnahmegruppe,
- Servicekonto,
- API-Berechtigung.

Solche CIs sind besonders wichtig für:

- Sicherheit,
- Change Enablement,
- Audits,
- Incident-Diagnose,
- Berechtigungsprobleme,
- und Compliance.

---

**CMDB-Reports**

Nützliche Reports können sein:

- aktive produktive Services,
- Services ohne Owner,
- CIs ohne Servicezuordnung,
- CIs ohne aktuelle Prüfung,
- ablaufende Zertifikate,
- veraltete Versionen,
- kritische CIs ohne Monitoring,
- Changes pro CI,
- Incidents pro CI,
- Known Errors pro Service,
- CIs mit fehlenden Beziehungen.

Reports sollen nicht nur Zahlen liefern.

Sie sollen konkrete Verbesserungen ermöglichen.

---

**CMDB-Einführung schrittweise beginnen**

Eine CMDB sollte nicht zwingend sofort alles abbilden.

Sinnvoller Start:

1. kritische Services auswählen
2. wichtigste CIs erfassen
3. zentrale Beziehungen dokumentieren
4. Owner festlegen
5. Change-Prozess anbinden
6. Datenqualität prüfen
7. Nutzung in Incident und Change etablieren
8. schrittweise erweitern

Eine CMDB scheitert häufig, wenn zu Beginn zu viel auf einmal erfasst werden soll.

---

**Minimal sinnvolle CMDB für kleine Umgebungen**

Für kleine Umgebungen kann auch eine einfache Struktur ausreichen.

Mögliche Mindestinformationen:

- Service,
- wichtige Komponenten,
- Owner,
- Umgebung,
- Status,
- Version,
- wichtigste Abhängigkeiten,
- Dokumentationslink,
- letzter Change,
- letzte Prüfung.

Das kann auch in einem einfachen ITSM-Tool, Wiki oder einer strukturierten Tabelle beginnen.

Wichtig ist Pflege und Nutzung.

---

**Governance der CMDB**

CMDB-Governance legt Regeln fest.

Zu klären ist:

- Wer ist für das CMDB-Modell verantwortlich?
- Wer darf CIs erstellen?
- Wer darf CIs ändern?
- Wer prüft Datenqualität?
- Welche Felder sind Pflicht?
- Wie werden Dubletten behandelt?
- Wie werden Changes eingebunden?
- Wie oft werden kritische CIs geprüft?
- Welche Reports werden genutzt?
- Wie werden Fehler korrigiert?

Ohne Governance wird eine CMDB schnell unzuverlässig.

---

### Rollen im CMDB-Betrieb

Mögliche Rollen:

| Rolle | Aufgabe |
|---|---|
| **Configuration Manager** | steuert Modell, Qualität und Regeln |
| **Service Owner** | verantwortet Serviceinformationen |
| **CI Owner** | verantwortet einzelne CIs |
| **Supportgruppe** | nutzt und ergänzt Betriebsinformationen |
| **Change Manager** | sorgt für Aktualisierung nach Changes |
| **Security Team** | nutzt Daten für Sicherheitsbewertung |
| **Asset Manager** | liefert Asset- und Vertragsdaten |

Die Rollen müssen nicht überall als eigene Stellen existieren.

Die Verantwortlichkeiten müssen aber klar sein.

---

### Praxisbeispiel: Incident mit CMDB-Nutzen

#### Situation

Mehrere Benutzer melden, dass das Mitarbeiterportal nicht erreichbar ist.

#### CMDB hilft durch:

- Servicezuordnung,
- betroffene Anwendung,

- Datenbankbeziehung,
- Zertifikatsbeziehung,
- letzter Change,
- zuständige Supportgruppe,
- Kritikalität des Services.

**Ergebnis**

Die Analyse konzentriert sich schneller auf Datenbank, Zertifikat und letzten Change.

---

**Praxisbeispiel: Change mit CMDB-Nutzen**

**Situation**

Ein Datenbankserver soll aktualisiert werden.

**CMDB zeigt:**

- drei Anwendungen nutzen diesen Server,
- eine Anwendung ist geschäftskritisch,
- ein Lieferant muss für Test erreichbar sein,
- Backupjob muss vor dem Change geprüft werden,
- Service Desk muss informiert werden.

**Ergebnis**

Der Change wird nicht als isoliertes Serverupdate behandelt, sondern als Serviceänderung mit Abhängigkeiten.

---

**Praxisbeispiel: Sicherheitslücke**

**Situation**

Eine kritische Schwachstelle betrifft eine bestimmte Softwareversion.

**CMDB hilft durch:**

- Liste betroffener Systeme,
- Servicezuordnung,
- Kritikalität,
- Owner,
- Umgebung,
- externe Erreichbarkeit,
- Patchstatus,
- Changes zur Behebung.

**Ergebnis**

Patchpriorisierung wird schneller und nachvollziehbarer.

---

**Praxisbeispiel: Abgelaufenes Zertifikat verhindern**

**Situation**

Mehrere Zertifikate wurden bisher nur in einzelnen Runbooks erwähnt.

**Problem**

Ablaufdaten wurden nicht zentral überwacht.

**Verbesserung**

Zertifikate werden als CIs erfasst mit:

- Ablaufdatum,
- Servicebezug,
- Owner,
- Monitoring,
- Erneuerungsprozess,
- Dokumentationslink.

**Nutzen**

Ausfälle durch abgelaufene Zertifikate werden besser vermeidbar.

---

**Typische Fehler**

**Fehler 1**

CMDB wird als reine Inventarliste aufgebaut.

---

**Fehler 2**

Zu viele CIs werden erfasst, aber nicht gepflegt.

---

**Fehler 3**

Services und Beziehungen fehlen.

---

**Fehler 4**

Owner sind nicht definiert.

---

**Fehler 5**

Daten werden nicht durch Changes aktualisiert.

---

**Fehler 6**

Discovery-Daten werden ungeprüft übernommen.

---

**Fehler 7**

Dubletten und uneinheitliche Namen werden nicht bereinigt.

---

**Fehler 8**

CMDB wird im Incident- und Change-Prozess nicht genutzt.

---

**Fehler 9**

Kritikalität wird technisch statt servicebezogen bewertet.

---

**Fehler 10**

Cloud-Ressourcen, Zertifikate und Schnittstellen fehlen.

---

**Fehler 11**

Reports zeigen Probleme, aber niemand verfolgt Verbesserungen.

---

**Fehler 12**

CMDB-Ziel und Pflegeaufwand passen nicht zusammen.

---

**Checkliste CMDB-Modell**

- [ ] Ziel der CMDB ist klar
- [ ] relevante Services sind definiert
- [ ] CI-Typen sind festgelegt

- [ ] Pflichtfelder sind definiert
- [ ] Beziehungstypen sind festgelegt
- [ ] Namenskonvention ist beschrieben
- [ ] Rollen und Verantwortlichkeiten sind klar
- [ ] Datenquellen sind bekannt
- [ ] Datenqualitätsregeln sind definiert
- [ ] Change-Prozess ist angebunden
- [ ] Reporting ist festgelegt
- [ ] regelmäßige Prüfung ist vorgesehen

---

**Checkliste CMDB-Eintrag**

- [ ] CI-Name eindeutig
- [ ] CI-Typ korrekt
- [ ] Status aktuell
- [ ] Umgebung korrekt
- [ ] Servicezuordnung vorhanden
- [ ] Owner oder Supportgruppe eingetragen
- [ ] Version oder Konfigurationsstand gepflegt
- [ ] Kritikalität nachvollziehbar
- [ ] Beziehungen gepflegt
- [ ] Dokumentationslink vorhanden
- [ ] letzte Änderung nachvollziehbar
- [ ] letzte Prüfung dokumentiert

---

**Checkliste Datenqualität**

- [ ] keine Dubletten vorhanden
- [ ] veraltete CIs archiviert
- [ ] produktive CIs besitzen Owner
- [ ] kritische CIs besitzen Servicebezug
- [ ] wichtige Beziehungen sind gepflegt
- [ ] Versionen sind aktuell
- [ ] Statuswerte sind plausibel
- [ ] Cloud-Ressourcen sind berücksichtigt

- [ ] Zertifikate sind erfasst
- [ ] Schnittstellen sind erfasst
- [ ] Changes aktualisieren CMDB-Daten
- [ ] Reports werden regelmäßig geprüft

---

**Checkliste CMDB im Change nutzen**

- [ ] betroffene CIs im Change verknüpft
- [ ] abhängige Services geprüft
- [ ] Owner identifiziert
- [ ] Kritikalität bewertet
- [ ] letzte Changes geprüft
- [ ] parallele Changes geprüft
- [ ] Dokumentationsbedarf erkannt
- [ ] CMDB-Aktualisierung nach Change geplant
- [ ] neue Beziehungen ergänzt
- [ ] entfernte Beziehungen bereinigt

---

**Bedeutung für Fachinformatiker für Systemintegration**

Fachinformatiker nutzen und pflegen CMDB-Daten im praktischen Betrieb.

Wichtig ist:

- Systeme eindeutig benennen,
- produktive und nicht produktive Umgebungen unterscheiden,
- Versionen und Konfigurationen sauber dokumentieren,
- Abhängigkeiten erkennen,
- Owner und Supportgruppen eintragen,
- Changes mit CIs verknüpfen,
- veraltete Daten melden,
- und CMDB-Daten bei Incidents, Problems und Changes aktiv nutzen.

Eine gute CMDB macht technische Arbeit nicht bürokratischer.

Sie macht sie nachvollziehbarer, sicherer und schneller.

---

**Zusammenfassung**

> Ziel der CMDB festlegen
> ↓
> relevante Services und CI-Typen bestimmen
> ↓
> Attribute und Beziehungen definieren
> ↓
> Datenquellen anbinden
> ↓
> Owner und Pflegeverantwortung klären
> ↓
> CIs und Beziehungen erfassen
> ↓
> CMDB in Incident, Problem und Change nutzen
> ↓
> Datenqualität regelmäßig prüfen
> ↓
> CMDB schrittweise verbessern

---

**Merksätze**

> Eine CMDB ist nur wertvoll, wenn ihre Daten genutzt und gepflegt werden.

> Beziehungen sind wichtiger als eine lange Liste einzelner Systeme.

> Eine CMDB ersetzt keine Betriebsdokumentation, sie verknüpft relevante Informationen.

> Discovery hilft bei Technik, ersetzt aber keine fachliche Verantwortung.

> Jede relevante Änderung sollte auch die CMDB aktualisieren.

>

> Schlechte CMDB-Daten können zu falschen Entscheidungen führen.

> “ Klein anfangen und konsequent pflegen ist besser als groß planen und nicht aktuell halten.

---

**Verwandte Seiten**

- 6.1 Service Configuration Management – Ziele, Begriffe und Grundlagen
- 6.2 Configuration Items und Beziehungen
- 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
- IT Asset 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:

- CMDB-Inhalte,
- Feldbeispiele,
- Datenqualitätsregeln,
- Rollen,
- Checklisten,
- Reports,
- und Praxisbeispiele

sind herstellerneutrale Praxisempfehlungen.

ITIL schreibt keine universelle:

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

für alle Organisationen vor.

Die konkrete Umsetzung muss an:

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

angepasst werden.

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

---