6.2 Configuration Items und Beziehungen

> **Kurz erklärt**
>
> Ein **Configuration Item (CI)** ist ein Bestandteil, der für einen IT-Service wichtig ist und deshalb verwaltet werden soll.
>
> Beziehungen zwischen CIs zeigen, wie Services, Anwendungen, Systeme, Netzwerke, Datenbanken, Zertifikate, Lieferanten und andere Bestandteile zusammenhängen.
>
> Der eigentliche Wert entsteht nicht nur durch die Liste einzelner CIs, sondern durch das Verständnis ihrer Abhängigkeiten.

Warum Configuration Items wichtig sind

IT-Services bestehen meistens aus mehreren technischen und organisatorischen Bestandteilen.

Beispiel:

Ein Mitarbeiterportal kann abhängig sein von:

- Webserver,
- Anwendung,
- Datenbank,
- Identity Provider,
- MFA,
- DNS,
- Zertifikat,
- Load Balancer,
- Firewall,
- Storage,
- Backup,
- Monitoring,
- Schnittstellen,
- Lieferant,
- Dokumentation,
- und Supportgruppe.

Wenn diese Bestandteile nicht bekannt sind, werden Incidents, Problems und Changes schwieriger.

Configuration Items helfen dabei, diese Bestandteile strukturiert zu erfassen.

Was ist ein Configuration Item?

Ein Configuration Item ist ein Element, das für die Bereitstellung, den Betrieb, die Steuerung oder die Unterstützung eines Service relevant ist.

Ein CI kann technisch oder nicht-technisch sein.

Beispiele:

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

Nicht jedes vorhandene Objekt muss automatisch ein CI sein.

Entscheidend ist, ob das Objekt für Servicebetrieb, Risiko, Support, Change oder Steuerung relevant ist.

CI ist nicht gleich Asset

Ein Asset ist häufig ein wirtschaftlich oder vertraglich relevantes Objekt.

Ein CI ist ein für den Servicebetrieb relevantes Objekt.

Asset	Configuration Item
Fokus auf Besitz, Kosten, Vertrag oder Lebenszyklus	Fokus auf Servicebeziehung, Abhängigkeit und Betrieb
wichtig für Einkauf und Lizenzierung	wichtig für Incident, Problem, Change und Release
Beispiel: Notebook mit Kaufdatum	Beispiel: Notebook als Teil eines verwalteten Arbeitsplatzservices

Asset	Configuration Item
Beispiel: Server als Anlagegut	Beispiel: Server als Bestandteil einer produktiven Anwendung

Ein Objekt kann gleichzeitig Asset und CI sein.

Beispiel:

Ein produktiver Datenbankserver ist ein Asset und gleichzeitig ein CI.

Welche Objekte sollten als CI erfasst werden?

Ein Objekt sollte eher als CI erfasst werden, wenn:

- ein Service davon abhängig ist,
- ein Ausfall Benutzer betrifft,
- Changes daran risikoreich sind,
- es für Sicherheitsbewertung wichtig ist,
- es für Incident-Diagnose benötigt wird,
- es eine relevante Version oder Konfiguration besitzt,
- es durch einen Lieferanten unterstützt wird,
- es eine wichtige Beziehung zu anderen Komponenten hat,
- es für Compliance, Audit oder Nachvollziehbarkeit wichtig ist,
- oder es in Problem- und Change-Analysen regelmäßig benötigt wird.

Ein Objekt muss nicht erfasst werden, nur weil es technisch existiert.

Die Pflege muss einen praktischen Nutzen haben.

Beispiele für CI-Typen

CI-Typ	Beispiel
Business Service	Mitarbeiterportal
Technical Service	Datenbankplattform
Anwendung	Warenwirtschaft
Server	PRD-APP-PORTAL-01
Datenbank	SQL-Cluster Produktion
Netzwerkgerät	Standort-Firewall
Cloud-Ressource	App Service, Storage Account
Sicherheit	TLS-Zertifikat, MFA-Richtlinie
Identität	Entra-ID-Gruppe

CI-Typ	Beispiel
Schnittstelle	REST-API zum ERP-System
Monitoring	Alarmregel für Zertifikatsablauf
Backup	Backupjob für Datenbank
Dokumentation	Betriebsrunbook
Lieferant	Supportvertrag Provider

Die CI-Typen sollten zur Organisation passen.

Zu viele CI-Typen können die Pflege unnötig erschweren.

Zu wenige CI-Typen können wichtige Unterschiede verdecken.

Service-CI und technische CIs unterscheiden

Ein Service selbst kann als CI betrachtet werden.

Beispiel:

- Service: Mitarbeiterportal
- Anwendung: Portal-App
- Server: Webserver
- Datenbank: Portal-DB
- Zertifikat: portal.example.tld
- Identity Provider: zentrale Anmeldung
- Supportgruppe: Application Support

Der Service ist das, was Benutzer wahrnehmen.

Die technischen CIs sind Bestandteile, die den Service ermöglichen.

Diese Unterscheidung hilft, technische Informationen mit Benutzer- und Geschäftsbezug zu verbinden.

Attribute eines Configuration Items

Ein CI sollte nur so viele Attribute besitzen, wie sinnvoll gepflegt und genutzt werden können.

Mögliche Attribute:

- CI-Name,
- CI-Typ,
- Beschreibung,

- Status,
- Umgebung,
- Version,
- Standort,
- Servicezuordnung,
- Owner,
- Supportgruppe,
- Lieferant,
- Kritikalität,
- Sicherheitsklassifizierung,
- Dokumentationslink,
- letzte Änderung,
- letzte Prüfung,
- Beziehungen,
- verknüpfte Incidents,
- verknüpfte Problems,
- verknüpfte Changes.

Wichtig ist nicht maximale Datenmenge.

Wichtig ist nutzbare Datenqualität.

CI-Name

Der CI-Name sollte eindeutig und verständlich sein.

Ungeeignet:

- Server1,
- AppNeu,
- TestAlt,
- FirewallFinal,
- Datenbank2.

Besser:

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

Eine Namenskonvention kann enthalten:

- Umgebung,
- Standort,

- Funktion,
- Servicebezug,
- laufende Nummer.

Die konkrete Namenskonvention muss zur Organisation passen.

CI-Status

Der Status zeigt, in welchem Zustand sich ein CI befindet.

Mögliche Status:

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

Der Status ist wichtig für:

- Incident-Diagnose,
- Change-Planung,
- Sicherheitsbewertung,
- Lizenz- und Asset-Abgleich,
- Außerbetriebnahme,
- und Dokumentationspflege.

Ein CI, das außer Betrieb ist, sollte nicht mehr als aktive Abhängigkeit eines produktiven Services erscheinen.

Umgebung

Die Umgebung zeigt, wo ein CI genutzt wird.

Beispiele:

- Entwicklung,
- Test,
- Integration,
- Abnahme,
- Staging,

- Produktion,
- Disaster Recovery,
- Labor,
- Schulung.

Ein Change an einem Testsystem ist anders zu bewerten als ein Change an einem produktiven System.

Fehlende Umgebungskennzeichnung kann gefährlich sein.

Beispiel:

Ein Administrator führt eine Änderung auf einem System aus, das für Test gehalten wird, tatsächlich aber produktiv ist.

Version und Konfiguration

Für viele CIs ist die Version wichtig.

Beispiele:

- Betriebssystemversion,
- Anwendungsversion,
- Datenbankversion,
- Firmwareversion,
- Clientversion,
- Konfigurationsstand,
- Zertifikatslaufzeit,
- API-Version,
- Treiberversion.

Versionen helfen bei:

- Incident-Diagnose,
- Known Errors,
- Sicherheitsbewertung,
- Patchmanagement,
- Release Management,
- Problem Management,
- und Change-Bewertung.

Beispiel:

Wenn nur VPN-Client-Version 5.8 betroffen ist, muss diese Information auffindbar sein.

Owner und Supportgruppe

Für wichtige CIs sollte klar sein:

- Wer ist fachlich verantwortlich?
- Wer ist technisch verantwortlich?
- Wer betreibt das CI?
- Wer unterstützt bei Incidents?
- Wer darf Changes genehmigen?
- Wer pflegt Dokumentation?
- Welcher Lieferant ist beteiligt?

Unklare Verantwortlichkeiten führen zu Verzögerungen.

Ungeeignet:

> Niemand weiß, wem dieser Server gehört.

Besser:

> Service Owner, technischer Owner, Supportgruppe und Lieferant sind dokumentiert.

Kritikalität

Die Kritikalität beschreibt, wie wichtig ein CI oder der zugehörige Service ist.

Mögliche Stufen:

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

Die Kritikalität beeinflusst:

- Incident-Priorisierung,
- Change-Risikobewertung,
- Monitoring,
- Backup-Anforderungen,
- Patchplanung,

- Notfallplanung,
- Lieferantensteuerung,
- und Service Reviews.

Wichtig:

Die Kritikalität sollte nicht nur technisch bewertet werden.

Ein unscheinbarer Server kann geschäftskritisch sein, wenn ein zentraler Prozess davon abhängt.

Sicherheitsklassifizierung

Manche CIs benötigen zusätzliche Sicherheitsinformationen.

Beispiele:

- enthält personenbezogene Daten,
- verarbeitet vertrauliche Daten,
- besitzt Internetzugang,
- ist extern erreichbar,
- besitzt Administratorzugänge,
- ist Teil kritischer Infrastruktur,
- ist auditrelevant,
- verarbeitet Zahlungsdaten,
- besitzt erhöhte Verfügbarkeitspflichten.

Solche Informationen helfen bei:

- Risikobewertung,
- Change Enablement,
- Sicherheitsvorfällen,
- Patchpriorisierung,
- Zugriffskontrolle,
- und Compliance.

Was sind CI-Beziehungen?

CI-Beziehungen beschreiben, wie Configuration Items miteinander verbunden sind.

Beispiele:

- Service nutzt Anwendung,
- Anwendung läuft auf Server,
- Server nutzt Storage,
- Anwendung nutzt Datenbank,

- Anwendung authentifiziert gegen Identity Provider,
- Webservice nutzt Zertifikat,
- Standort nutzt Firewall,
- Anwendung kommuniziert mit API,
- Monitoring überwacht Service,
- Backupjob sichert Datenbank,
- Lieferant unterstützt System.

Beziehungen machen sichtbar, welche Auswirkungen ein Ausfall oder Change haben kann.

Warum Beziehungen wichtiger sind als Einzelinformationen

Eine Liste einzelner CIs beantwortet nur:

> Was gibt es?

Beziehungen beantworten:

> Was hängt wovon ab?

Beispiel:

CI-Liste:

- APP-01,
- DB-01,
- FW-01,
- CERT-01.

Beziehungen:

- Mitarbeiterportal läuft auf APP-01.
- APP-01 nutzt DB-01.
- APP-01 ist über FW-01 erreichbar.
- Mitarbeiterportal nutzt CERT-01.
- DB-01 nutzt Storage-Cluster PRD-STO-01.

Erst durch Beziehungen wird klar, welche Komponenten gemeinsam einen Service ermöglichen.

Typische Beziehungstypen

Beziehung	Bedeutung
nutzt	Ein CI verwendet ein anderes CI
läuft auf	Anwendung läuft auf Server oder Plattform
ist abhängig von	Service funktioniert nur mit anderem CI
verbindet zu	Netzwerk- oder Schnittstellenbeziehung
wird überwacht von	Monitoringbeziehung
wird gesichert durch	Backupbeziehung
wird unterstützt von	Lieferanten- oder Supportbeziehung
ist Teil von	Komponente gehört zu größerem Service
ersetzt	neues CI ersetzt altes CI
kommuniziert mit	technische Schnittstelle zwischen Systemen

Die Beziehungsarten sollten verständlich und einheitlich genutzt werden.

Beispiel: Mitarbeiterportal

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

Weitere Beziehungen:

- Mitarbeiterportal nutzt DNS-Eintrag portal.example.tld.
- Mitarbeiterportal nutzt TLS-Zertifikat PRD-CERT-PORTAL.
- Mitarbeiterportal nutzt Identity Provider.
- Webserver wird durch Monitoring überwacht.
- Datenbank wird durch Backupjob gesichert.
- Anwendung wird durch Application Support betreut.

Beispiel: VPN-Service

```
VPN-Service
   ↓ nutzt
VPN-Gateway
   ↓ nutzt
Firewall
   ↓ nutzt
Internetanbindung
```

Zusätzliche Abhängigkeiten:

- VPN-Client,
- Zertifikat,
- MFA,
- Identity Provider,
- Benutzergruppen,
- DNS,
- Monitoring,
- Netzwerkteam,
- Lieferant.

Wenn MFA ausfällt, kann der VPN-Service betroffen sein, obwohl das VPN-Gateway technisch läuft.

Beispiel: Druckservice

```
Druckservice
   ↓ nutzt
Druckserver
   ↓ verwaltet
Netzwerkdrucker
   ↓ nutzt
Netzwerksegment
```

Weitere Abhängigkeiten:

- Druckertreiber,
- Benutzerberechtigungen,
- Spooler-Dienst,
- Papier- und Verbrauchsmaterial,
- Standortnetz,
- Fachanwendung,
- Etikettenvorlagen.

Ein Druckproblem kann also viele Ursachen haben.

Ohne Beziehungen wird oft nur der Drucker selbst betrachtet.

Beispiel: Schnittstelle zwischen Anwendungen

```
Anwendung A
   ↓ sendet Daten an
API-Gateway
   ↓ leitet weiter an
Anwendung B
   ↓ schreibt in
Datenbank B
```

Zusätzliche Abhängigkeiten:

- Zertifikat,
- Firewall-Regel,
- DNS,
- API-Token,
- Berechtigungen,
- Monitoring,
- Lieferant,
- Datenformat.

Eine kleine Änderung am API-Token kann den gesamten Datenaustausch stoppen.

Beziehungen und Incident Management

Bei Incidents helfen Beziehungen, die Auswirkung zu verstehen.

Beispiele:

- Welcher Service ist betroffen?
- Welche CIs sind beteiligt?
- Gibt es eine gemeinsame Abhängigkeit?
- Welche anderen Services könnten betroffen sein?
- Wer ist zuständig?
- Welche Changes wurden zuletzt an beteiligten CIs durchgeführt?
- Gibt es Known Errors für diese Version oder Komponente?

Beispiel:

Mehrere Anwendungen melden Fehler.

CI-Beziehungen zeigen, dass alle denselben Datenbankcluster nutzen.

Die Diagnose wird schneller eingegrenzt.

Beziehungen und Problem Management

Problem Management nutzt CI-Beziehungen, um Muster zu erkennen.

Beispiele:

- wiederkehrende Incidents betreffen dieselbe Hardwaremodellserie,
- mehrere Services nutzen dasselbe Zertifikat,
- mehrere Anwendungen sind von derselben Schnittstelle abhängig,
- ein Standort hat Probleme mit derselben Firewall,
- eine bestimmte Clientversion erzeugt viele Tickets.

Ohne Beziehungen bleiben solche Muster oft unsichtbar.

Beziehungen und Change Enablement

Bei Changes sind Beziehungen besonders wichtig.

Zu prüfen ist:

- Welche CIs werden geändert?
- Welche Services hängen davon ab?
- Welche Benutzergruppen sind betroffen?
- Welche Schnittstellen werden beeinflusst?
- Welche anderen Teams müssen beteiligt werden?
- Welche Dokumentation muss aktualisiert werden?
- Welche Tests sind notwendig?
- Welche Rollback-Auswirkungen gibt es?

Ein Change an einem kleinen CI kann große Auswirkungen haben, wenn viele Services davon abhängen.

Beispiel: Change-Auswirkung durch Beziehung

Change

Ein Zertifikat soll erneuert werden.

Ohne Beziehung

Es wird nur der Webserver betrachtet.

Mit Beziehung

Es wird sichtbar:

- Zertifikat wird vom Mitarbeiterportal genutzt,
- Anmeldung läuft über Identity Provider,
- externe Schnittstelle prüft Zertifikatskette,
- Monitoring muss nach Erneuerung geprüft werden,
- Service Desk muss mögliche Anmeldefehler kennen.

Dadurch wird der Change besser geplant.

Beziehungen und Release Management

Release Management benötigt Beziehungen, um zu verstehen:

- welche Komponenten Bestandteil des Releases sind,
- welche Zielumgebungen betroffen sind,
- welche Abhängigkeiten bestehen,
- welche Versionen zusammenpassen,
- welche Benutzergruppen betroffen sind,
- welche Knowledge-Artikel aktualisiert werden müssen.

Ein Release kann fehlschlagen, wenn eine abhängige Komponente vergessen wird.

Beziehungen und Information Security Management

Sicherheitsarbeit benötigt CI-Beziehungen.

Beispiele:

- Welche Services sind extern erreichbar?
- Welche Systeme verarbeiten vertrauliche Daten?
- Welche Anwendungen nutzen ein gefährdetes Zertifikat?
- Welche Server verwenden eine verwundbare Softwareversion?
- Welche Benutzergruppen haben Zugriff?
- Welche Schnittstellen übertragen sensible Daten?
- Welche Systeme sind von einem kompromittierten Konto betroffen?

Ohne Beziehungen ist Sicherheitsbewertung unvollständig.

Beziehungen und Notfallplanung

Für Notfallplanung sind Abhängigkeiten entscheidend.

Zu klären ist:

- Welche Services sind geschäftskritisch?
- Welche CIs werden für Wiederherstellung benötigt?
- Welche Reihenfolge ist beim Wiederanlauf notwendig?
- Welche Datenbanken müssen zuerst verfügbar sein?
- Welche Netzwerkverbindungen sind erforderlich?
- Welche Lieferanten müssen erreichbar sein?
- Welche Backupjobs gehören zu welchem Service?
- Welche Dokumentation wird im Notfall benötigt?

Ein Wiederanlaufplan ohne Abhängigkeiten ist häufig unvollständig.

Tiefe der Beziehungspflege

Nicht jede Beziehung muss bis ins kleinste Detail gepflegt werden.

Zu klären ist:

- Welche Beziehungen werden wirklich genutzt?
- Welche Beziehungen helfen bei Incident, Problem und Change?
- Welche Services sind kritisch?
- Welche CIs ändern sich häufig?
- Welche Daten können automatisiert gepflegt werden?
- Welche Daten müssen manuell gepflegt werden?

Zu wenig Beziehungspflege macht die CMDB nutzlos.

Zu viel Detailtiefe macht sie schwer wartbar.

Beispiel für sinnvolle Detailtiefe

Für einen geschäftskritischen Service sinnvoll:

- Service,
- Anwendung,
- Datenbank,
- Server,
- Netzwerk,
- Zertifikat,
- Identity Provider,
- Backup,
- Monitoring,

- Owner,
- Lieferant,
- wichtigste Schnittstellen.

Für einen einfachen Testservice kann weniger Detail genügen.

Die Pflege muss zum Risiko passen.

Horizontale und vertikale Beziehungen

Vertikale Beziehungen

zeigen den Aufbau eines Service von oben nach unten.

Beispiel:

```
Service
   ↓
Anwendung
   ↓
Server
   ↓
Plattform
   ↓
Storage
```

Horizontale Beziehungen

zeigen Verbindungen zwischen gleichartigen oder verbundenen Komponenten.

Beispiel:

```
Anwendung A ↔ API ↔ Anwendung B
```

Beide Beziehungstypen sind wichtig.

Vertikale Beziehungen helfen bei Auswirkungsanalyse.

Horizontale Beziehungen helfen bei Schnittstellen- und Kommunikationsproblemen.

Upstream und Downstream

Bei Schnittstellen wird oft zwischen Upstream und Downstream unterschieden.

Begriff	Bedeutung
Upstream	vorgelagertes System, von dem Daten oder Dienste kommen
Downstream	nachgelagertes System, das Daten oder Dienste erhält

Beispiel:

```
CRM-System
   ↓ sendet Kundendaten an
ERP-System
   ↓ sendet Rechnungsdaten an
Buchhaltungssystem
```

Wenn das CRM-System fehlerhafte Daten liefert, können downstream weitere Probleme entstehen.

Single Point of Failure erkennen

CI-Beziehungen können Single Points of Failure sichtbar machen.

Beispiel:

Mehrere Services nutzen denselben Datenbankserver.

Wenn dieser Server ausfällt, sind alle Services betroffen.

Mögliche Maßnahmen:

- Hochverfügbarkeit prüfen,
- Backup und Restore testen,
- Monitoring verbessern,
- Kapazität planen,
- Change-Risiko erhöhen,
- Notfallverfahren dokumentieren.

Beziehungen helfen also nicht nur bei Dokumentation, sondern auch bei Risikobewertung.

Dubletten und Namensprobleme

CI-Beziehungen werden unbrauchbar, wenn CIs mehrfach oder uneinheitlich erfasst sind.

Beispiele:

- APP-01,
- app01,
- AppServer Portal,
- PRD-APP-PORTAL-01

beschreiben möglicherweise dasselbe System.

Folgen:

- Incidents werden falsch verknüpft,
- Changes wirken unvollständig,
- Beziehungen fehlen,
- Reports werden ungenau,
- Owner erscheinen widersprüchlich.

Namenskonventionen und Dublettenprüfung sind deshalb wichtig.

Veraltete Beziehungen

Beziehungen können veralten durch:

- Migrationen,
- Releases,
- Servertausch,
- Cloud-Umzug,
- neue Schnittstellen,
- Außerbetriebnahme,
- Notfalländerungen,
- Lieferantenwechsel,
- Prozessänderungen.

Veraltete Beziehungen sind riskant.

Beispiel:

Ein Change wird als unkritisch bewertet, weil eine alte Beziehung fehlt.

Nach Umsetzung fällt ein abhängiger Service aus.

Beziehungen durch Changes aktualisieren

Jeder relevante Change sollte prüfen:

- Wurde ein CI geändert?
- Wurde ein CI neu erstellt?
- Wurde ein CI entfernt?

- Hat sich eine Version geändert?
- Hat sich eine Beziehung geändert?
- Hat sich ein Owner geändert?
- Hat sich eine Schnittstelle geändert?
- Hat sich ein Zertifikat geändert?
- Muss die CMDB aktualisiert werden?

Configuration Management darf nicht getrennt von Change Enablement betrachtet werden.

Beziehungen durch Discovery erkennen

Technische Beziehungen können teilweise automatisch erkannt werden.

Beispiele:

- Server kommuniziert mit Datenbank,
- Anwendung öffnet Verbindung zu API,
- Zertifikat ist auf Webserver installiert,
- Softwareversion ist vorhanden,
- Netzwerkgerät ist erreichbar,
- Cloud-Ressource existiert.

Aber:

Discovery erkennt nicht immer fachliche Bedeutung.

Ein Tool kann eine Verbindung sehen.

Es weiß nicht automatisch, ob diese Verbindung geschäftskritisch ist.

Manuell gepflegte Beziehungen

Manuell gepflegt werden oft:

- Service Owner,
- fachliche Kritikalität,
- Geschäftsprozessbezug,
- Supportmodell,
- Lieferantenverantwortung,
- Risikoakzeptanz,
- geplante Ablösung,
- Dokumentationslinks,
- fachliche Schnittstellenbedeutung.

Diese Informationen brauchen Verantwortliche und regelmäßige Prüfung.

Beziehungen visualisieren

CI-Beziehungen können als Diagramm, Tabelle oder Abhängigkeitsansicht dargestellt werden.

Mögliche Formen:

- Service Map,
- Abhängigkeitsdiagramm,
- Schnittstellenübersicht,
- Tabellenansicht,
- Netzwerkplan,
- CMDB-Beziehungsansicht,
- Architekturdiagramm.

Wichtig ist, dass die Darstellung für den Zweck geeignet ist.

Ein Management-Review benötigt andere Details als eine technische Fehleranalyse.

Service Map

Eine Service Map zeigt die wichtigsten Bestandteile eines Services.

Beispiel:

```
Service: Mitarbeiterportal

Benutzer
   ↓
DNS
   ↓
Load Balancer
   ↓
Webserver
   ↓
Anwendung
   ↓
Datenbank
   ↓
Storage

Zusätzlich:
Identity Provider, Zertifikat, Monitoring, Backup, Supportgruppe
```

Eine Service Map muss nicht jedes Detail enthalten.

Sie soll die wichtigsten Abhängigkeiten verständlich machen.

Tabellarische Beziehungspflege

Eine einfache Tabelle kann für kleinere Umgebungen ausreichend sein.

Service	CI	Beziehung	Abhängig von	Owner
Mitarbeiterportal	Portal-App	läuft auf	PRD-APP-PORTAL-01	App-Team
Mitarbeiterportal	Portal-App	nutzt	PRD-DB-PORTAL-01	DB-Team
Mitarbeiterportal	Portal-App	nutzt	Identity Provider	IAM-Team
Mitarbeiterportal	Webservice	nutzt	TLS-Zertifikat	Plattform-Team

Wichtig ist, dass die Tabelle gepflegt und genutzt wird.

CI-Beziehungen in Tickets nutzen

Tickets sollten relevante CIs enthalten.

Das hilft bei:

- Auswirkungsanalyse,
- Priorisierung,
- Eskalation,
- Trendanalyse,
- Problem Management,
- Change-Verknüpfung,
- Reporting.

Beispiel:

Ein Incident wird nicht nur als „Anwendung gestört“ erfasst.

Er wird mit Service, Anwendungsversion, Datenbank und betroffener Schnittstelle verknüpft.

Dadurch können spätere Muster erkannt werden.

CI-Beziehungen in Changes nutzen

Ein Change Request sollte betroffene CIs enthalten.

Beispiele:

- Zielserver,
- Anwendung,
- Datenbank,
- Zertifikat,
- Firewall-Regel,
- Schnittstelle,
- Monitoringregel,
- Dokumentation.

Zusätzlich sollten abhängige Services geprüft werden.

Ein Change an einer zentralen Komponente muss anders bewertet werden als ein Change an einem isolierten Testsystem.

CI-Beziehungen in Problems nutzen

Ein Problem Record sollte relevante CIs und Beziehungen enthalten.

Beispiel:

Mehrere Incidents betreffen verschiedene Anwendungen.

Problem Management erkennt:

- alle Anwendungen nutzen dieselbe API,
- API nutzt ein ablaufendes Zertifikat,
- Zertifikatsmonitoring fehlt.

Die CI-Beziehung hilft, eine gemeinsame Ursache zu finden.

Praxisbeispiel: Datenbank als gemeinsame Abhängigkeit

Situation

Drei Anwendungen melden Performance-Probleme.

Einzelbetrachtung

Jedes Team sucht zunächst in seiner Anwendung.

CI-Beziehungen zeigen

- alle drei Anwendungen nutzen denselben Datenbankcluster,
- Datenbankcluster nutzt denselben Storage,
- Storage zeigt hohe Latenz.

Nutzen

Die Analyse wird schneller auf die gemeinsame Abhängigkeit gelenkt.

Praxisbeispiel: Zertifikat als CI

Situation

Anmeldung am Service schlägt fehl.

CI-Beziehungen zeigen

- Service nutzt Webzertifikat,
- Zertifikat läuft heute ab,
- Identity Provider prüft Zertifikatskette,
- Monitoring überwacht nur HTTP-Status, nicht Zertifikatslaufzeit.

Verbesserung

Zertifikate werden künftig als eigene CIs mit Ablaufdatum, Owner und Monitoringbeziehung gepflegt.

Praxisbeispiel: Standort-Firewall

Situation

Ein Standort meldet Zugriffsstörungen.

CI-Beziehungen zeigen

- Standort nutzt bestimmte Firewall,
- mehrere Services laufen über dieselbe VPN-Verbindung,
- letzte Änderung war eine Firewall-Regel,
- betroffene Benutzergruppe sitzt nur an diesem Standort.

Nutzen

Die Fehlersuche wird auf Standortnetz, Firewall und VPN eingegrenzt.

Praxisbeispiel: Veraltete Beziehung

Situation

Eine alte Datenbank wird abgeschaltet.

Problem

In der CMDB ist nicht dokumentiert, dass ein Reporting-Service noch darauf zugreift.

Folge

Nach Abschaltung fällt das Reporting aus.

Lerneffekt

Vor Außerbetriebnahme müssen Abhängigkeiten geprüft und alte Beziehungen bereinigt werden.

Typische Fehler

Fehler 1

Nur technische Geräte werden als CIs betrachtet.

Fehler 2

Services selbst werden nicht als CIs erfasst.

Fehler 3

Beziehungen zwischen CIs fehlen.

Fehler 4

Beziehungen werden einmal erstellt und danach nicht gepflegt.

Fehler 5

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

Fehler 6

CIs haben keinen Owner.

Fehler 7

CI-Namen sind uneinheitlich oder doppelt.

Fehler 8

Discovery-Daten werden ohne fachliche Prüfung übernommen.

Fehler 9

Changes aktualisieren CI-Beziehungen nicht.

Fehler 10

Tickets werden nicht mit betroffenen CIs verknüpft.

Fehler 11

Kritikalität wird ohne Servicebezug bewertet.

Fehler 12

Schnittstellen, Zertifikate und Cloud-Ressourcen werden vergessen.

Checkliste: Ist ein Objekt ein sinnvolles CI?

- [ ] unterstützt es einen Service?
- [ ] kann ein Ausfall Benutzer betreffen?
- [ ] ist es für Incidents relevant?
- [ ] ist es für Problems relevant?
- [ ] ist es für Changes relevant?
- [ ] besitzt es wichtige Beziehungen?
- [ ] besitzt es relevante Versionen oder Konfigurationen?
- [ ] hat es Sicherheits- oder Compliance-Bedeutung?
- [ ] braucht es einen Owner?
- [ ] muss es regelmäßig geprüft werden?
- [ ] lohnt sich der Pflegeaufwand?

Checkliste CI-Attribute

- [ ] eindeutiger Name
- [ ] CI-Typ

- [ ] Beschreibung
- [ ] Status
- [ ] Umgebung
- [ ] Version oder Konfigurationsstand
- [ ] betroffener Service
- [ ] Owner
- [ ] Supportgruppe
- [ ] Lieferant, falls relevant
- [ ] Kritikalität
- [ ] Sicherheitsklassifizierung, falls relevant
- [ ] Dokumentationslink
- [ ] letzte Prüfung
- [ ] relevante Beziehungen

Checkliste CI-Beziehungen

- [ ] Service zu Anwendung verknüpft
- [ ] Anwendung zu Server oder Plattform verknüpft
- [ ] Anwendung zu Datenbank verknüpft
- [ ] Anwendung zu Schnittstellen verknüpft
- [ ] DNS-Abhängigkeit geprüft
- [ ] Zertifikate verknüpft
- [ ] Identity Provider oder MFA verknüpft
- [ ] Netzwerk und Firewall berücksichtigt
- [ ] Storage berücksichtigt
- [ ] Backup berücksichtigt
- [ ] Monitoring berücksichtigt
- [ ] Lieferant berücksichtigt
- [ ] Supportgruppe verknüpft
- [ ] Dokumentation verknüpft

Checkliste Beziehungspflege nach Changes

- [ ] neue CIs angelegt
- [ ] entfernte CIs archiviert

- [ ] geänderte Versionen aktualisiert
- [ ] geänderte Schnittstellen dokumentiert
- [ ] neue Abhängigkeiten ergänzt
- [ ] entfernte Abhängigkeiten gelöscht
- [ ] Owner aktualisiert
- [ ] Lieferantenbezug aktualisiert
- [ ] Dokumentationslinks geprüft
- [ ] Monitoring- und Backupbeziehungen geprüft
- [ ] Service Map aktualisiert
- [ ] Ticket oder Change Record verknüpft

Bedeutung für Fachinformatiker für Systemintegration

Fachinformatiker arbeiten praktisch mit vielen Configuration Items.

Beispiele:

- Server,
- Clients,
- Firewalls,
- Switches,
- VPN,
- Drucker,
- Benutzergruppen,
- Zertifikate,
- Backupjobs,
- Monitoringregeln,
- Cloud-Ressourcen,
- Anwendungen,
- Schnittstellen.

Wichtig ist nicht nur, diese Komponenten technisch zu kennen.

Wichtig ist auch zu verstehen:

- welcher Service davon abhängt,
- welche Benutzer betroffen wären,
- welche Changes riskant sind,
- welche Beziehungen bei Fehleranalyse helfen,
- wer zuständig ist,
- und welche Informationen aktuell gehalten werden müssen.

Gute CI-Beziehungen machen technische Arbeit schneller, sicherer und nachvollziehbarer.

Zusammenfassung

> wichtige Services identifizieren
> ↓
> relevante Configuration Items bestimmen
> ↓
> sinnvolle Attribute erfassen
> ↓
> Owner und Kritikalität festlegen
> ↓
> Beziehungen zwischen CIs dokumentieren
> ↓
> Serviceabhängigkeiten sichtbar machen
> ↓
> CIs in Incidents, Problems und Changes nutzen
> ↓
> Beziehungen nach Changes aktualisieren
> ↓
> Datenqualität regelmäßig prüfen

Merksätze

> Ein CI ist nur dann wertvoll, wenn es für den Servicebetrieb relevant ist.

> Beziehungen zeigen, welche Auswirkungen ein Ausfall oder Change haben kann.

> Eine CI-Liste ohne Beziehungen ist nur begrenzt hilfreich.

> Technisch kleine Komponenten können geschäftlich sehr wichtig sein.

> Discovery erkennt Technik, aber nicht automatisch fachliche Bedeutung.

>

> CI-Beziehungen müssen nach Changes gepflegt werden.

> “ Gute Configuration-Daten verbessern Incident, Problem, Change und Security Management.

Verwandte Seiten

- 6.1 Service Configuration Management – Ziele, Begriffe und Grundlagen
- 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,
- Beziehungstypen,
- Service-Maps,
- Tabellen,
- Checklisten,
- und Praxisbeispiele

sind herstellerneutrale Praxisempfehlungen.

ITIL schreibt keine universelle:

- CI-Typenliste,
- Beziehungstypenliste,
- Attributliste,
- Service-Map-Struktur,
- Namenskonvention,
- oder konkrete Modellierungstiefe

für alle Organisationen vor.

Die konkrete Umsetzung muss an:

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

angepasst werden.

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