FH AACHEN
UNIVERSITY OF APPLIED SCIENCES

Fachhochschule Aachen
Campus Jülich

# Detailliertes Infrastrukturkonzept zur Entwicklung einer Configuration Management Database (CMDB)

**Seminararbeit**

**Luiz Post**
**Matr. No.: 3651388**
**Erstbetreuer:Prof. Dr. rer. nat. Alexander Voß**
**Zweitbetreuer: B.Sc Andre Kapp**

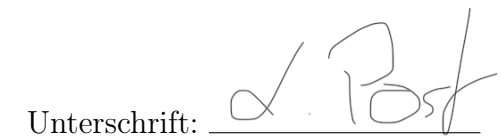
**Fachbereich 9**
**Medizintechnik und Technomathematik**
**Angewandte Mathematik und Informatik B.Sc.**

**Aachen, Dezember 2025**

# Eidesstattliche Erklärung

Hiermit versichere ich, dass ich die Seminararbeit mit dem Thema **„Detailliertes Infrastrukturkonzept zur Entwicklung einer Configuration Management Database (CMDB)“** selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Alle Ausführungen, die anderen Schriften wörtlich oder sinngemäß entnommen wurden, sind kenntlich gemacht. Die Arbeit wurde in gleicher oder ähnlicher Form noch nicht als Studien- oder Prüfungsleistung eingereicht.
Im Rahmen der Erstellung dieser Arbeit wurde das KI-System „GPT-5.1“ unterstützend zur sprachlichen Überarbeitung sowie zur fachlichen Reflexion und Präzisierung eigenständig entwickelter Argumente genutzt. Eine Übernahme von KI-generierten Texten oder inhaltlichen Lösungsvorschlägen erfolgte nicht. Sämtliche fachlichen Aussagen, Bewertungen und Schlussfolgerungen wurden eigenständig erarbeitet und verantwortet. Die Nutzung erfolgte im Einklang mit der Zweckbestimmung des Systems sowie unter Beachtung datenschutz- und urheberrechtlicher Vorgaben.

Ich verpflichte mich, ein Exemplar der Seminararbeit fünf Jahre aufzubewahren und es auf Verlangen dem Prüfungsamt des Fachbereichs Medizintechnik und Technomathematik auszuhändigen.

Ort, Datum: Aachen, den 15.12.2025 Unterschrift: 

**Zusammenfassung**

Die vorliegende Arbeit befasst sich mit der Konzeption eines technischen Gesamtarchitekturmodells einschließlich der zugrunde liegenden Infrastruktur einer Configuration Management Database (CMDB) auf Basis einer modernen Open-Source-Technologie namens *Datagerry*. Ziel der Untersuchung war es, eine skalierbare und modular erweiterbare Architektur zu entwickeln, die den Anforderungen an Transparenz, Automatisierung und Integrationsfähigkeit in bestehende IT-Umgebungen gerecht wird. Als zentrale Plattform wurde das Open-Source-Tool *DataGerry* ausgewählt und einem systematischen Feasibility-Check unterzogen, um seine Eignung als CMDB zu bewerten. Darauf aufbauend entstand ein Architekturkonzept, das die für eine CMDB erforderlichen Kernkomponenten – Datenmodellierung, API-Zugriffsschicht und Verwaltungsoberfläche – strukturiert beschreibt und ihre Rolle innerhalb der Systemlandschaft definiert. Die Implementierung der automatisierten Bereitstellung erfolgte mittels *Infrastructure as Code* unter Verwendung von Ansible und containerisierten Deployments über Docker Compose. Zudem wurde die Integration in CI/CD-Pipelines und Monitoring-Systeme konzeptionell berücksichtigt, um nach der Entwicklung einen durchgängigen Automatisierungsprozess sicherzustellen. Die Ergebnisse zeigen, dass die untersuchten Technologien grundsätzlich das Potenzial bieten, eine flexible, wartbare und erweiterbare Grundlage für ein CMDB-System zu bilden. Die Arbeit liefert hierfür erste konzeptionelle Ansätze, die in zukünftigen Projekten weiter ausgearbeitet und praktisch validiert werden müssen. Perspektivisch ergeben sich insbesondere Ansatzpunkte für technische Implementierungen, Security-Härtung sowie den Ausbau automatisierter Schnittstellen.

# Inhaltsverzeichnis

# 1 Einleitung

## 1.1 Motivation und Relevanz des Themas

In modernen Unternehmensumgebungen gewinnt die effiziente Verwaltung von IT-Infrastrukturen zunehmend an Bedeutung. Mit der stetig wachsenden Anzahl an Servern, Containern, Cloud-Diensten und Applikationen nimmt die Komplexität der Systemlandschaft deutlich zu. Unternehmen stehen dabei vor der Herausforderung, ihre Systeme nicht nur zu betreiben, sondern auch deren Konfigurationen, Abhängigkeiten und Zustände transparent zu dokumentieren und nachzuvollziehen.

Eine Configuration Management Database (CMDB) stellt in diesem Zusammenhang ein zentrales Werkzeug dar, um sämtliche Konfigurationsobjekte (Configuration Items, CIs) einer IT-Landschaft systematisch zu erfassen und deren Beziehungen zueinander zu modellieren. Sie bildet damit die Grundlage für Prozesse wie Change Management, Incident Management und Problem Management im Sinne des ITIL-Frameworks[1].

Gleichzeitig gewinnen Open-Source-Technologien an Bedeutung, da sie neben der Kostenersparnis vor allem durch ihre Anpassbarkeit und Transparenz überzeugen. Durch frei zugängliche Quelltexte lassen sich Sicherheitsaspekte nachvollziehbar prüfen, Integrationen über offene Schnittstellen leichter realisieren und individuelle Erweiterungen ohne Bindung an proprietäre Anbieter umsetzen. Werkzeuge wie *DataGerry* zeigen, dass ein transparenter und erweiterbarer Ansatz zur Verwaltung von Infrastrukturkonfigurationen auch ohne kommerzielle Lizenzmodelle realisierbar ist. Neben DataGerry existieren auch etablierte Open-Source-CMDBs wie i-doit oder iTop sowie Asset-Tools wie Snipe-IT. Die Wahl fiel bewusst auf DataGerry, da dessen generisches Datenmodell eine besonders flexible Objektdefinition ermöglicht[1]. Die Integration solcher Lösungen kann wesentlich zur Effizienzsteigerung und Standardisierung von IT-Prozessen beitragen.

## 1.2 Problemstellung: Fehlende Transparenz und Nachvollziehbarkeit in IT-Infrastrukturen

Laut der „State of the Nation Survey Findings – CMS/CMDB“ der Georg-August-Universität Göttingen besitzen lediglich etwa 42% der befragten Unternehmen eine implementierte CMDB; rund 24% befinden sich noch in der Entwicklung, 18% haben keine CMDB oder planen derzeit keine [2]. Es fehlt also trotz der Verfügbarkeit moderner Tools und DevOps-Ansätze wie *Infrastructure as Code*, *Continous Delivery* oder automatisierter *Service Discovery*, in vielen Organisationen an einer klar strukturierten und einheitlichen

[1] ITIL (Information Technology Infrastructure Library) ist ein weltweit anerkanntes Rahmenwerk für das IT-Service-Management (ITSM). Es bietet eine Sammlung von Best Practices, Prozessen und Konzepten zur Planung, Bereitstellung und Unterstützung von IT-Services mit dem Ziel, Effizienz, Qualität und Kundenzufriedenheit zu verbessern.

Dokumentation der IT-Komponenten. Informationen über Systeme, Dienste, Abhängigkeiten und Verantwortlichkeiten liegen oft verteilt in verschiedenen Abteilungen, Tools oder gar nur in individueller Kenntnis einzelner Mitarbeiter vor.

Diese Fragmentierung führt zu Intransparenz, erschwert Fehleranalysen und verzögert Entscheidungen bei Änderungen oder Störungen im IT-Betrieb. Besonders in Umgebungen, in denen DevOps- und Automatisierungsstrategien eingesetzt werden, kann das Fehlen einer zentralen und aktuellen Konfigurationsdatenbasis zu erheblichen betrieblichen Risiken führen – etwa durch falsche oder veraltete Konfigurationsobjekte, fehlerhafte Abhängigkeitsmodelle oder Relationen zwischen Systemen. Solche Inkonsistenzen können wiederum zu längeren Ausfallzeiten, Fehlentscheidungen im Change-Management und unerwarteten Störungen im Produktivbetrieb führen.

Vor diesem Hintergrund besteht die Notwendigkeit, ein ganzheitliches Konzept zu entwickeln, das sowohl die technische Architektur als auch die organisatorischen Anforderungen einer modernen CMDB abbildet. Dieses Konzept muss auf offenen Standards beruhen und Integrationsmöglichkeiten in bestehende Systeme wie CI/CD-Pipelines oder Monitoring-Umgebungen berücksichtigen.

## 1.3 Zielsetzung der Arbeit

Ziel dieser Arbeit ist die Entwicklung eines detaillierten Infrastrukturkonzepts für den Aufbau einer Configuration Management Database (CMDB). Dieses Konzept soll die funktionalen und nicht-funktionalen Anforderungen an eine CMDB erfassen, eine geeignete Systemarchitektur vorschlagen und mögliche Technologien sowie Integrationspunkte aufzeigen.

Besonderes Augenmerk liegt dabei auf der Modularität der Architektur sowie der Möglichkeit, diese durch offene Schnittstellen, erweiterbare Datenmodelle und flexible Integrationsmechanismen problemlos an bestehende IT-Landschaften anzubinden und bei Bedarf um zusätzliche Funktionen zu erweitern. Zudem soll das Konzept anhand existierender Open-Source-Lösungen – insbesondere DataGerry – untersucht werden, um praxisnahe Ansätze sowie mögliche Verbesserungspotenziale zu identifizieren. Die Analyse umfasst dabei insbesondere das Datenmodell, die verfügbaren APIs, Import- und Exportmechanismen, Aspekte der Skalierbarkeit, Authentifizierungsverfahren sowie die vorhandene Historisierung2.

Durch die konzeptionelle Ausarbeitung soll aufgezeigt werden, wie eine strukturierte und gut integrierbare CMDB-Architektur grundsätzlich zur Erhöhung von Transparenz und Effizienz in der IT-Infrastrukturverwaltung beitragen kann.

## 1.4 Abgrenzung und Vorgehensweise

Die vorliegende Arbeit konzentriert sich auf die konzeptionelle Ausarbeitung der für eine CMDB relevanten infrastrukturellen Komponenten. Dazu gehören insbesondere Überlegungen zur Containerisierung, Netzwerk- und Service-Architektur, Datenhaltung, Authentifizierungsmechanismen sowie möglichen Monitoring-Integrationen. Es werden keine vollständigen Implementierungen oder produktiven Deployments durchgeführt; vielmehr werden die technischen und methodischen Grundlagen beschrieben, die als Basis für eine spätere praktische Umsetzung dienen können. Anschließend erfolgt eine detaillierte Anforderungsanalyse, in der konkrete Use-cases erarbeitet werden, die als Basis für das Architekturdesign dient. Darauf aufbauend wird ein Infrastrukturkonzept vorgestellt, das den Aufbau und Betrieb einer modularen, containerisierten CMDB beschreibt.

Abschließend werden Integrationsmöglichkeiten und Automatisierungsstrategien mit gängigen CI/CD- und Monitoring-Tools diskutiert. Die Arbeit schließt mit einer Zusammenfassung der Ergebnisse und einem Ausblick auf die weiterführende Entwicklung der CMDB.

# 2 Theoretische und technische Grundlagen

## 2.1 Begriff des Configuration Managements

Configuration Management (CM) beschreibt den systematischen Prozess zur Erfassung, Dokumentation und Steuerung der Konfigurationselemente (Configuration Items, CIs) einer IT-Infrastruktur über deren gesamten Lebenszyklus hinweg. Ziel des Configuration Managements ist es, sicherzustellen, dass die Integrität und Nachvollziehbarkeit der Systemkonfigurationen zu jedem Zeitpunkt gewährleistet ist. Moderne CMDBs können prinzipiell über APIs automatisch aktualisiert werden, sofern entsprechende Discovery- oder Integrationsprozesse vorhanden sind.

Nach ITIL[3] umfasst das Configuration Management die Identifikation, Kontrolle, Statusüberwachung und Verifizierung aller CIs, die zur Erbringung von IT-Services beitragen. Zu den typischen Configuration Items zählen Hardwarekomponenten, Softwareversionen, Netzwerkelemente, virtuelle Maschinen, Container und sogar Dokumentationen. Diese werden in einer zentralen Datenbank – der sogenannten Configuration Management Database (CMDB) – verwaltet, um den aktuellen Zustand der IT-Landschaft konsistent und transparent abzubilden.

Durch die Einführung eines strukturierten Configuration Managements können Fehlerquellen reduziert, Änderungen besser nachvollzogen und Risiken im IT-Betrieb minimiert werden. Insbesondere in dynamischen Infrastrukturen, in denen Cloud- und Container-Technologien eingesetzt werden, gewinnt ein automatisiertes und skalierbares Configuration Management zunehmend an Bedeutung.

## 2.2 Rolle und Bedeutung einer CMDB im IT-Service-Management

Die CMDB stellt das zentrale Informationssystem im IT-Service-Management (ITSM) dar. Sie dient als Datenquelle für zahlreiche operative und strategische Prozesse, darunter Change Management, Incident Management, Problem Management, Asset Management und Service Level Management.

Ihre Hauptaufgabe besteht darin, die Beziehungen und Abhängigkeiten zwischen den verschiedenen Configuration Items abzubilden. Diese Beziehungen ermöglichen eine gezielte Analyse von Auswirkungen bei Systemänderungen oder Störungen. Beispielsweise kann bei einem Ausfall eines Servers sofort ermittelt werden, welche Applikationen, Benutzer oder Geschäftsprozesse davon betroffen sind.

Darüber hinaus unterstützt eine CMDB die Einhaltung regulatorischer Anforderungen und interner Compliance-Vorgaben, indem sie eine nachvollziehbare Historie von Änderungen und Zuständen bereitstellt. In größeren Organisationen dient sie zudem als Grundlage für Audits und Sicherheitsprüfungen.

In einem modernen ITSM-Kontext wird die CMDB zunehmend in automatisierte Pro-

zesse integriert. Über Schnittstellen zu Monitoring-, Deployment- und Orchestrierungswerkzeugen kann sie aktuelle Konfigurationsdaten automatisch erfassen und aktualisieren. Dadurch entwickelt sich die CMDB von einem rein dokumentarischen System zu einem aktiven Bestandteil der operativen IT-Steuerung.

## 2.3 ITIL-Referenzmodell und Best Practices

Das *Information Technology Infrastructure Library* (ITIL 4)-Framework definiert Best Practices für das moderne IT-Service-Management und beschreibt anhand seiner Practices und Value Streams[2], wie Organisationen ihre Services ganzheitlich planen, bereitstellen und kontinuierlich verbessern können. Innerhalb dieses Frameworks nimmt das Service Configuration Management eine zentrale Rolle ein.

In ITIL 4 bildet die Configuration Management Database (CMDB) einen wesentlichen Bestandteil dieser Practice. Sie stellt sicher, dass Informationen über Konfigurationselemente (CIs) und deren Beziehungen konsistent verwaltet werden. Die CMDB unterstützt dabei andere ITIL-4-Practices wie *Change Enablement*, indem sie die Bewertung potenzieller Auswirkungen geplanter Änderungen erleichtert, sowie das *Incident Management*, das durch aktuelle CI-Daten Störungen schneller analysieren und beheben kann.

ITIL 4 empfiehlt, den Aufbau einer CMDB schrittweise und bedarfsgerecht vorzunehmen. Ein überdimensioniertes oder zu detailliertes Modell erhöht das Risiko hoher Pflegekosten, mangelnder Datenqualität und unübersichtlicher Strukturen. Stattdessen wird ein pragmatischer Ansatz empfohlen, bei dem zunächst die wichtigsten Services, CIs und Abhängigkeiten modelliert und anschließend sukzessive erweitert werden.

Neben ITIL 4 existieren weitere Frameworks wie COBIT oder ISO/IEC 20000, die ebenfalls Leitlinien für das Configuration Management bereitstellen. Diese Standards betonen die Bedeutung einer konsistenten, nachvollziehbaren und transparenten Konfigurationsverwaltung als Grundlage für ein effektives IT-Service-Management.

## 2.4 Open-Source-Ansätze im Configuration Management

Mit dem zunehmenden Einsatz von Open-Source-Software in Unternehmensumgebungen haben sich auch im Bereich des Configuration Managements zahlreiche quelloffene Werkzeuge etabliert. Bekannte Vertreter sind etwa *Ansible*, *Puppet*, *Chef* und *SaltStack*, die primär auf die Automatisierung von Infrastrukturkonfigurationen ausgerichtet sind.

Im Gegensatz zu diesen Tools, die operative Konfigurationsänderungen durchführen, fokussieren sich Open-Source-CMDB-Systeme wie *i-doit*, *OCS Inventory NG* oder *DataGerry* auf die zentrale Erfassung und Verwaltung von Konfigurationsdaten. Diese Systeme

[2] In ITIL4 bezeichnet ein Value Stream die Gesamtheit aller Schritte und Aktivitäten, die notwendig sind, um einen Service oder ein Produkt von der Anforderung bis zur Wertschöpfung für den Kunden bereitzustellen.

bieten eine flexible Datenstruktur, um individuelle Objekttypen und Abhängigkeiten zu modellieren.

Der Einsatz von Open-Source-Lösungen bringt dabei mehrere Vorteile mit sich: Unternehmen vermeiden Vendor Lock-ins[3] , können Anpassungen an eigene Anforderungen vornehmen und profitieren von einer aktiven Entwickler-Community. Darüber hinaus ermöglichen offene Schnittstellen (REST-APIs) eine einfache Integration in bestehende DevOps- und ITSM-Umgebungen.

Die Kombination von Automatisierungswerkzeugen und Open-Source-CMDBs bietet somit eine leistungsfähige Basis für moderne, dynamische IT-Infrastrukturen, in denen Transparenz, Skalierbarkeit und Nachvollziehbarkeit zentrale Anforderungen darstellen.

## 2.5 Einordnung und Architektur von DataGerry

*DataGerry* ist ein Open-Source-CMDB-System, das speziell darauf ausgelegt ist, eine flexible und anpassbare Verwaltung von Configuration Items zu ermöglichen. Im Gegensatz zu klassischen CMDB-Lösungen, die häufig starre Datenmodelle verwenden, verfolgt DataGerry einen generischen Ansatz, bei dem Benutzer eigene Objekttypen, Attribute und Relationen definieren können.

Die Architektur von DataGerry basiert auf einem modularen Aufbau. Es wird *RabbitMQ* als Mesagging-System benutzt und im Backend werden die Daten in einer *MongoDB*-NoSQL-Datenbank gespeichert, die eine hohe Flexibilität und Skalierbarkeit bei der Verwaltung unstrukturierter Daten ermöglicht. Über eine REST-API können externe Systeme auf die gespeicherten Konfigurationsdaten zugreifen, was eine Integration mit Monitoring-Systemen, CI/CD-Pipelines oder Inventarisierungstools erleichtert.

Das Frontend von DataGerry ist webbasiert und ermöglicht eine intuitive Verwaltung der Datenmodelle, Konfigurationsobjekte und Beziehungen. Die Anwendung ist in Python implementiert und lässt sich durch containerisierte Bereitstellung (z. B. via Docker oder Podman) leicht in bestehende Infrastrukturen integrieren.

Im Kontext dieser Arbeit dient DataGerry als Referenzsystem, um die konzeptionellen Überlegungen einer modernen, offenen CMDB zu konkretisieren. Dabei wird insbesondere untersucht, wie sich eine skalierbare Architektur, offene Schnittstellen und Automatisierungskonzepte in ein integriertes Infrastrukturdesign einfügen lassen[1].

[3]Der Begriff *Vendor Lock-in* bezeichnet die Abhängigkeit eines Unternehmens von einem bestimmten Anbieter (Vendor) aufgrund proprietärer Technologien, Formate oder Schnittstellen. Diese Bindung erschwert oder verhindert den Wechsel zu alternativen Produkten oder Anbietern, da hohe Umstellungs- und Integrationskosten entstehen können. Im Kontext von IT-Infrastrukturen bedeutet ein Vendor Lock-in, dass Organisationen bei Software, Cloud-Diensten oder Hardware an einen Hersteller gebunden sind, was Flexibilität und Innovationsfähigkeit einschränken kann [4, 5].

# 3 Anforderungsanalyse

## 3.1 Methodik der Anforderungsanalyse

Die Anforderungsanalyse stellt einen zentralen Bestandteil bei der Konzeption und Entwicklung von Softwaresystemen dar. Sie dient der systematischen Erhebung, Strukturierung und Dokumentation aller Anforderungen, die an das zu entwickelnde System gestellt werden . Im Kontext dieser Arbeit bildet die Anforderungsanalyse die Grundlage für die spätere Konzeption einer Configuration Management Database (CMDB). Dabei werden die methodischen Schritte beschrieben, die typischerweise bei der Ermittlung von Anforderungen zum Einsatz kommen, ohne dass eine vollständige empirische Erhebung durchgeführt wird.

### Vorgehensweise bei der Anforderungserhebung

Zur Ermittlung von Anforderungen existieren verschiedene bewährte Methoden, die je nach Projektumfang und Zielsetzung kombiniert werden können. Zu den klassischen Ansätzen zählen insbesondere:

- **Interviews:** Gespräche mit relevanten Stakeholdern – wie IT-Administratoren, DevOps-Ingenieuren oder IT-Service-Managern – dienen dazu, ein Verständnis für bestehende Prozesse, Probleme und Erwartungen zu entwickeln. Durch halbstrukturierte Interviews lassen sich sowohl explizite Anforderungen als auch implizite Bedürfnisse erfassen [6].

- **Dokumentenanalyse:** Die Untersuchung vorhandener Dokumentationen (z. B. Systemhandbücher, Netzwerkpläne, ITIL-Prozessbeschreibungen) ermöglicht es, bestehende Strukturen und Schnittstellen zu identifizieren. Für eine CMDB sind insbesondere Informationen über vorhandene Konfigurationsobjekte und ihre Abhängigkeiten von Bedeutung.

- **Workshops und Use-Case-Analysen:** In gemeinsamen Workshops mit Stakeholdern können Anforderungen konsolidiert und priorisiert werden. Die Modellierung von Use Cases hilft, Systemfunktionen aus Anwendersicht zu verstehen und konkrete Nutzungsszenarien zu definieren [7].

Für das vorliegende konzeptionelle Projekt wird ein hypothetisches Vorgehen angenommen, das sich an diesen Methoden orientiert. Ziel ist es, eine konsistente und nachvollziehbare Sammlung von Anforderungen zu erstellen, die als Grundlage für das Systemdesign dienen kann.

#### Klassifizierung der Anforderungen

Die ermittelten Anforderungen werden üblicherweise in **funktionale** und **nicht-funktionale Anforderungen** unterteilt [8]:

- **Funktionale Anforderungen** beschreiben, *was* das System leisten soll – also konkrete Funktionen, Prozesse oder Verhaltensweisen. Im Falle einer CMDB betrifft dies etwa die Verwaltung von Configuration Items (CIs), die Darstellung von Abhängigkeiten, Such- und Filterfunktionen oder Schnittstellen zu anderen IT-Systemen.
- **Nicht-funktionale Anforderungen** legen fest, *wie* das System seine Aufgaben erfüllen soll. Dazu zählen unter anderem Anforderungen an Sicherheit, Verfügbarkeit, Skalierbarkeit, Performance und Wartbarkeit. Für eine CMDB ist beispielsweise sicherzustellen, dass die Datenintegrität gewährleistet ist, sensible Systeminformationen geschützt werden und die Lösung in wachsenden IT-Umgebungen performant bleibt.

#### Zusammenfassung und Ausblick

Die in diesem Abschnitt beschriebenen methodischen Schritte bilden die Grundlage für die nachfolgende Spezifikation der Anforderungen. Während hier der Fokus auf der allgemeinen Methodik der Anforderungserhebung und -klassifizierung lag, werden in den folgenden Abschnitten (**3.2** und **3.3**) die konkreten funktionalen und nicht-funktionalen Anforderungen an die zu konzipierende CMDB im Detail beschrieben.

## 3.2 Funktionale Anforderungen an eine CMDB

Die funktionalen Anforderungen definieren die zentralen Aufgaben und Eigenschaften, die eine Configuration Management Database (CMDB) erfüllen muss, um ihre Kernfunktionen im Rahmen des IT-Service-Managements (ITSM) nach ITIL effektiv zu unterstützen. Diese Anforderungen ergeben sich aus der Analyse der ITIL-Prozesse, den betrieblichen Zielsetzungen sowie den typischen Herausforderungen moderner IT-Infrastrukturen.

#### Erfassung und Verwaltung von Configuration Items (CIs)

Eine der primären Funktionen einer CMDB besteht in der strukturierten Erfassung und Verwaltung von *Configuration Items (CIs)*. Hierbei handelt es sich um sämtliche Komponenten einer IT-Umgebung, die für den Betrieb und die Verwaltung relevant sind – beispielsweise Server, virtuelle Maschinen, Container, Netzwerkelemente, Softwarepakete oder Cloud-Ressourcen [3, 9]. Jedes CI muss eindeutig identifizierbar sein und über Attribute wie Name, Typ, Status, Verantwortlicher und Version verfügen. Die CMDB sollte dabei die Möglichkeit bieten, unterschiedliche CI-Typen zu modellieren und benutzerdefinierte Attribute zu ergänzen, um anwendungsspezifische Informationen abzubilden.

### Beziehungsmanagement und Abhängigkeitsmodellierung

Neben der isolierten Erfassung einzelner Objekte ist die Abbildung von Beziehungen zwischen den CIs ein zentraler Bestandteil einer CMDB. Diese Beziehungen ermöglichen eine ganzheitliche Sicht auf die IT-Landschaft und bilden die Grundlage für Impact-Analysen im Change- und Incident-Management. Ein funktionales CMDB-System muss daher Relationstypen wie „läuft auf“, „abhängig von“ oder „gehört zu“ unterstützen. Dies erlaubt die Visualisierung von Infrastrukturabhängigkeiten und fördert das Verständnis komplexer Systemzusammenhänge [3, 1].

### Änderungs- und Versionshistorie

Zur Sicherstellung der Nachvollziehbarkeit ist die Versionierung von Konfigurationen und Änderungen unerlässlich. Eine CMDB soll in der Lage sein, Veränderungen an CIs automatisch oder manuell zu protokollieren und frühere Zustände wiederherstellbar zu machen. Dies bildet die Grundlage für eine revisionssichere Dokumentation und unterstützt das Problem- und Change-Management bei der Ursachenanalyse von Störungen. Open-Source-Systeme wie *DataGerry* bieten hierfür bereits modulare Audit-Mechanismen, die auf bestehenden Datenbankstrukturen (z. B. MongoDB) aufbauen [1].

### Schnittstellen und Integrationen

Da die CMDB häufig als zentrales Informationssystem in einer heterogenen IT-Landschaft fungiert, sind offene Schnittstellen (z. B. REST- oder GraphQL-APIs) eine zentrale funktionale Anforderung. Diese ermöglichen die Integration mit Monitoring-Tools (z. B. Prometheus, Zabbix), ITSM-Systemen (z. B. OTRS, ServiceNow) oder CI/CD-Pipelines (Jenkins). Dadurch können Konfigurationsdaten automatisiert aktualisiert, neue CIs erkannt und Änderungen in Echtzeit bzw. *"near real-time"* synchronisiert werden [10, 11].

### Such-, Filter- und Reporting-Funktionalität

Eine CMDB muss leistungsfähige Mechanismen zur Suche, Filterung und Auswertung der gespeicherten Daten bereitstellen. Hierzu zählen einfache Textsuchen, Attribut-basierte Filter sowie erweiterte Query-Funktionen, die beispielsweise alle CIs eines bestimmten Typs oder Status ausgeben können. Zusätzlich sind Funktionen zur Erstellung von Reports oder Dashboards erforderlich, um Management-Entscheidungen datenbasiert zu unterstützen [12].

### Benutzer- und Rechtemanagement

Für den produktiven Einsatz in größeren Organisationen ist ein fein granuliertes Benutzer- und Rechtemanagement notwendig. Dabei müssen unterschiedliche Rollen (z. B. Adminis-

trator, Operator, Auditor) mit jeweils spezifischen Berechtigungen ausgestattet werden können. Diese Zugriffskontrollen gewährleisten sowohl die Informationssicherheit als auch die Einhaltung von Compliance-Richtlinien [3].

#### Zusammenfassung

Die funktionalen Anforderungen einer CMDB lassen sich somit in fünf Kernbereiche unterteilen:

1. Erfassung und Verwaltung von CIs
2. Modellierung von Beziehungen und Abhängigkeiten
3. Änderungs- und Versionsmanagement
4. Schnittstellenintegration und Automatisierung
5. Benutzer- und Zugriffskontrolle

Diese Anforderungen bilden die Grundlage für das in Kapitel 4 vorgestellte Architekturkonzept und dienen als Basis für die spätere Evaluierung der Lösung.

## 3.3 Nicht-funktionale Anforderungen (Sicherheit, Skalierbarkeit, Wartbarkeit)

Neben den funktionalen Anforderungen spielen nicht-funktionale Anforderungen eine zentrale Rolle bei der Konzeption einer Configuration Management Database (CMDB). Sie definieren die Qualitätsmerkmale des Systems und bestimmen, in welchem Maße die funktionalen Anforderungen unter realen Betriebsbedingungen erfüllt werden können. Zu den wichtigsten nicht-funktionalen Aspekten zählen Sicherheit, Skalierbarkeit, Wartbarkeit, Verfügbarkeit und Performance.

#### Sicherheitsanforderungen

Die CMDB enthält zentrale Informationen über die gesamte IT-Infrastruktur und stellt somit ein besonders schützenswertes System dar. Sicherheitsanforderungen umfassen daher sowohl technische als auch organisatorische Maßnahmen. Zu den technischen Anforderungen zählen Authentifizierungs- und Autorisierungsmechanismen, verschlüsselte Datenübertragung (z. B. HTTPS/TLS), rollenbasiertes Zugriffskonzept und Audit-Logging [13, 11]. Darüber hinaus müssen Datenschutzanforderungen nach geltenden Normen (z. B. DSGVO[4] ) berücksichtigt werden, insbesondere wenn personenbezogene Daten oder Zu-

[4]Die Datenschutz-Grundverordnung (DSGVO) ist eine EU-Verordnung, die seit dem 25. Mai 2018 gilt und den Schutz personenbezogener Daten sowie den freien Datenverkehr innerhalb der Europäischen Union regelt [14].

ordnungen zu Benutzern gespeichert werden. Eine sichere CMDB implementiert daher „Privacy by Design“ und „Least Privilege“-Prinzipien.

### Skalierbarkeit und Performance

Da moderne IT-Landschaften stark dynamisch und oft hybrid aufgebaut sind, muss eine CMDB in der Lage sein, große Datenmengen effizient zu verarbeiten und bei Bedarf horizontal oder vertikal zu skalieren. Eine containerisierte Architektur – etwa basierend auf Podman oder Kubernetes – ermöglicht eine flexible Skalierung von Services wie Datenbank, API und Frontend [15]. Auch das zugrunde liegende Datenmodell muss so gestaltet sein, dass Lese- und Schreiboperationen performant bleiben, selbst bei mehreren zehntausend Configuration Items (CIs). NoSQL-Datenbanken wie MongoDB eignen sich hierfür besonders gut, da sie durch horizontale Skalierung und flexible Schemata eine hohe Anpassungsfähigkeit bieten [16].

### Wartbarkeit und Erweiterbarkeit

Eine CMDB ist ein langfristiges Kernsystem im IT-Betrieb. Ihre Architektur muss daher so ausgelegt sein, dass sie auch über Jahre hinweg wartbar und erweiterbar bleibt. Hierzu zählen eine saubere Code- und API-Struktur, modularer Aufbau der Komponenten sowie automatisierte Tests und Deployment-Prozesse (z. B. via CI/CD-Pipelines) [10]. Durch die Nutzung von Open-Source-Frameworks und Containerisierung kann die CMDB zudem einfacher aktualisiert und erweitert werden, ohne dass der laufende Betrieb gestört wird. Dies fördert die Nachhaltigkeit der IT-Systemlandschaft.

### Verfügbarkeit und Zuverlässigkeit

Da die CMDB ein zentrales Informationssystem ist, sollte sie eine hohe Verfügbarkeit gewährleisten. Dies kann durch den Einsatz von Redundanzmechanismen (z. B. Replikation der Datenbank), Load-Balancing und Monitoring realisiert werden. Eine Integration mit Überwachungssystemen wie Prometheus oder Zabbix ermöglicht frühzeitige Erkennung von Anomalien und trägt zur Stabilität des Betriebs bei [12]. Darüber hinaus sollten Backup- und Wiederherstellungsstrategien fester Bestandteil des Betriebskonzepts sein, um Datenverlust und Ausfallzeiten zu minimieren.

### Benutzerfreundlichkeit und Transparenz

Ein weiterer Qualitätsaspekt betrifft die Benutzerfreundlichkeit der CMDB. Nur wenn die Benutzeroberfläche intuitiv gestaltet und die Daten klar strukturiert dargestellt sind, kann das System im Alltag effektiv genutzt werden. Ziel ist eine transparente Informationsdarstellung, die sowohl technischen als auch organisatorischen Stakeholdern den Zugriff auf

relevante Daten ermöglicht. Dashboards, visuelle Beziehungsdarstellungen und kontextbasierte Filter tragen wesentlich zur Effizienz im Betrieb bei [1].

#### Zusammenfassung

Nicht-funktionale Anforderungen bilden die Grundlage für die langfristige Stabilität, Sicherheit und Leistungsfähigkeit einer CMDB. Sie ergänzen die funktionalen Anforderungen und definieren die qualitativen Rahmenbedingungen, unter denen die in Kapitel 4 vorgestellte Architektur betrieben werden soll.Eine erfolgreiche Implementierung berücksichtigt daher sowohl technische als auch organisatorische Aspekte, um eine nachhaltige, sichere und erweiterbare Lösung zu gewährleisten. Außerdem bilden alle Anforderungen(sowohl Funktionale als auch nicht-Funktionale) die Grundlage für den *Feasability-Check* auf den sich in Kapitel 4.5 bezogen wird. Dieser soll prüfen ob das Open Source-Tool *Datagerry* allen Anforderungen die wir in Kapitel 3.2 und 3.3 erarbeitet haben standhält.

## 3.4 Konkrete Use Cases für die CMDB-Implementierung

Um die im Rahmen der Anforderungsanalyse ermittelten funktionalen und nicht-funktionalen Anforderungen praxisnah zu validieren, werden in diesem Abschnitt konkrete Use Cases definiert. Diese dienen als exemplarische Szenarien, anhand derer das spätere Architekturkonzept (vgl. Kapitel 4) entwickelt werden kann.

Die ausgewählten Use Cases orientieren sich an typischen Herausforderungen im IT-Infrastrukturmanagement und spiegeln die Kernprozesse einer Configuration Management Database (CMDB) wieder. Dabei wird ein besonderer Fokus auf Aspekte wie Datenintegration, Automatisierung und Transparenz gelegt.

#### Use Case 1: Automatisierte Erfassung von Infrastrukturkomponenten

Ein zentraler Anwendungsfall einer CMDB besteht in der automatisierten Erfassung von physischen und virtuellen Infrastrukturkomponenten. In modernen, containerisierten Umgebungen werden Systeme häufig dynamisch bereitgestellt und wieder entfernt. Der Use Case beschreibt die automatische Erkennung und Synchronisation von neuen Servern, Containern oder Netzwerkgeräten mit der CMDB.

**Ziel:** Sicherstellung, dass die CMDB jederzeit den aktuellen Zustand der Infrastruktur abbildet. **Akteure:** Systemadministrator, Monitoring-System, Discovery-Agent. Der Discovery-Prozess erfolgt nicht nativ. Er müsste über Skripte oder externe Tools wie OCS Inventory NG implementiert werden. **Ablauf:**

1. Ein neues System wird in einer Cloud- oder Containerumgebung bereitgestellt.
2. Ein Discovery-Agent erfasst relevante Metadaten (z. B. Hostname, IP-Adresse, Betriebssystemversion).

3. Die Daten werden über eine API an die CMDB (DataGerry) übermittelt und dort als *Configuration Item (CI)* gespeichert.

4. Änderungen werden versioniert, um historische Zustände nachvollziehbar zu machen.

**Erwartetes Ergebnis:** Eine konsistente und stets aktuelle Datenbasis, die als Grundlage für Monitoring, Incident- und Change-Management dient [17].

#### Use Case 2: Integration mit Monitoring- und CI/CD-Systemen

Dieser Use Case beschreibt ein mögliches Integrationsszenario, in dem die CMDB in bestehende IT-Operations-Prozesse eingebunden wird. Da für gängige Monitoring-Lösungen wie Prometheus oder Zabbix sowie für CI/CD-Systeme wie Jenkins oder GitLab CI keine offiziellen oder vorkonfigurierten Integrationen existieren, erfolgt der Datenaustausch ausschließlich über individuell entwickelte Skripte oder API-Aufrufe.

**Ziel:** Verbesserung der Transparenz und Automatisierung, indem Konfigurationsdaten für Build-, Deployment- und Überwachungsprozesse zentral verfügbar sind.

**Akteure:** DevOps-Engineer, CI/CD-System, Monitoring-System. **Ablauf:**

1. Die CMDB stellt Informationen zu Services und Abhängigkeiten über eine REST-API bereit.

2. CI/CD-Jobs können diese Daten abrufen, um Deployment-Umgebungen dynamisch zu konfigurieren.

3. Monitoring-Systeme nutzen dieselben Daten, um automatisch neue Services zu erfassen oder Alarme zu konfigurieren.

**Erwartetes Ergebnis:** Reduzierter manueller Aufwand bei der Systemintegration sowie verbesserte Nachvollziehbarkeit von Änderungen in der Produktionsumgebung [18].

#### Use Case 3: Änderungsmanagement und Nachvollziehbarkeit

Ein wesentlicher Aspekt jeder CMDB ist die Unterstützung des Change Managements im Sinne des ITIL-Frameworks. Dieser Use Case beschreibt die Nachvollziehbarkeit von Konfigurationsänderungen und deren Auswirkungen auf abhängige Systeme.

**Ziel:** Minimierung von Risiken bei Änderungen durch eine klare Versionshistorie und Abhängigkeitsanalyse.

**Akteure:** Change Manager, IT-Service-Desk, CMDB-System. **Ablauf:**

1. Ein geplanter Change wird im ITSM-System (z. B. ServiceNow) erfasst.

2. Das CMDB-System identifiziert alle betroffenen CIs und zeigt deren Beziehungen an.

3. Nach Durchführung des Changes wird der neue Zustand automatisch dokumentiert.

**Erwartetes Ergebnis:** Erhöhte Transparenz und verbesserte Entscheidungsgrundlage bei geplanten Änderungen.

**Zusammenfassung der Use Cases** Die drei beschriebenen Use Cases bilden die Basis für die konzeptionelle und technologische Ausarbeitung der CMDB in Kapitel 4. Sie decken zentrale Funktionsbereiche ab:

- **Erfassung und Pflege** von *Configuration Items*(CIs) (Use Case 1),
- **Integration** in bestehende Betriebsprozesse (Use Case 2),
- **Transparenz und Kontrolle** im Änderungsmanagement (Use Case 3).

# 4 Konzeptionelle Architektur einer CMDB

Die konzeptionelle Architektur bildet das zentrale Ergebnis der vorliegenden Arbeit. Sie beschreibt die grundlegenden Systemkomponenten, deren logische Zusammenhänge sowie die Kommunikationsflüsse innerhalb der geplanten Configuration Management Database (CMDB). Ziel ist es, eine skalierbare, modular aufgebaute und erweiterbare Systemlandschaft zu entwerfen, die auf offenen Standards und Open-Source-Technologien basiert.

## 4.1 Überblick über die Zielarchitektur

Die in dieser Arbeit konzipierte Architektur folgt einem *modularen Aufbau*, bei dem einzelne Funktionsbereiche in klar abgegrenzten Systemkomponenten realisiert sind. Diese Modularität erlaubt es, Komponenten unabhängig voneinander zu entwickeln, zu warten und bei Bedarf zu erweitern. Die logische Struktur der Architektur ist in Abbildung 1 im Anhang dargestellt. Sie zeigt die wesentlichen Systemelemente und in welchen Netzwerken sie deployt werden sollen. Datenflüsse und Integrationspunkte zu bestehenden IT-Systemen sind ebenfalls abgebildet.

Das Gesamtsystem gliedert sich in drei Hauptschichten:

- **Präsentationsschicht:** Verantwortlich für die Benutzerinteraktion über Weboberflächen und APIs. Dazu gehören das Frontend (Svelte-basiert) und die DataGerry Admin UI.
- **Applikations- und Logikschicht:** In dieser Schicht befinden sich zentrale Dienste wie das BFF - Backend For Frontend (*Java/Quarkus*) sowie die DataGerry-API, die als Schnittstelle zur Datenhaltung fungiert.

- **Datenhaltungsschicht:** Besondere Aufmerksamkeit erfordert dabei die Sicherstellung der Datenkonsistenz sowie der referenziellen Beziehungen zwischen den gespeicherten Konfigurationsobjekten, insbesondere im Hinblick auf Hochverfügbarkeitsszenarien. Obwohl Docker Compose benutzt wurde vgl. Unterkapitel 5.3.2, müssen Mechanismen zur Replikation und Konsistenzwahrung der MongoDB-Datenbank gesondert berücksichtigt werden, da Docker Compose keine native HA-Unterstützung bietet

Ergänzt wird die Architektur durch unterstützende Komponenten, wie den *Keycloak*-Cluster zur Authentifizierung und Autorisierung, sowie externe Systeme (z. B. *Confluence*, *GitHub*, *CyberArk*, *Jenkins*, *Prometheus*), die über standardisierte Schnittstellen eingebunden werden; für sicherheitskritische Anwendungen wie die Secret-Verwaltung durch CyberArk wäre jedoch ein klar definiertes, abgesichertes Integrationskonzept erforderlich, etwa über rollenbasierte Zugriffskontrollen, Token-Verwaltung und den Einsatz kurzlebiger Anmeldeinformationen.[19]

## 4.2 Modularer Aufbau und Schnittstellen

Der modulare Aufbau gewährleistet eine klare Trennung der Verantwortlichkeiten und minimiert Abhängigkeiten zwischen den Komponenten. Die Hauptmodule der Architektur lassen sich wie folgt beschreiben:

1. **DataGerry Core:** Stellt die zentrale Plattform zur Verwaltung und Strukturierung von Konfigurationsdaten bereit. Über die REST-API werden alle Operationen (CRUD) auf den Configuration Items ausgeführt.

2. **BFF – Backend For Frontend:** Realisiert in *Java/Quarkus* und fungiert als Vermittler zwischen Frontend, DataGerry und externen Systemen wie *Confluence* (Dokumentation), *GitHub* (Quellcode-Verwaltung), *CyberArk* (Secret Management), *Jenkins* (CI/CD-Pipeline) und *Prometheus* (monitoring-Tool).

3. **Frontend:** Eine Svelte-basierte Benutzeroberfläche, die Nutzern einen intuitiven Zugriff auf die CMDB-Daten ermöglicht.

4. **Authentifizierung und Zugriffskontrolle:** Der *Keycloak*-Cluster (RHBK) übernimmt zentrale Funktionen der Benutzerverwaltung und Single-Sign-On-Integration.

5. **Persistenzschicht:** basiert auf einer *MongoDB*-Instanz, in der DataGerry die CMDB-Objekte sowie die zugehörigen Metadaten ablegt. Beziehungen zwischen Konfigurationselementen und Änderungsinformationen werden durch die Anwendung selbst verwaltet und als strukturierte Dokumente in MongoDB gespeichert, da MongoDB keine native referentielle Integrität oder Versionierung bereitstellt.

Durch diesen modularen Aufbau ist die Architektur flexibel erweiterbar und kann zukünftig um zusätzliche Module, wie etwa ein automatisiertes Discovery- oder Monitoring-Modul, ergänzt werden.

#### Begründung des BFFs und des Frontends

Für die Zielarchitektur wird ein Backend-for-Frontend (BFF) eingesetzt, um Funktionen bereitzustellen, die über die Möglichkeiten der reinen DataGerry-API hinausgehen. Da DataGerry architekturbedingt dem Repo-first-Ansatz folgt – das heißt, das Datenmodell wird primär in der MongoDB definiert und die REST-API anschließend als Zugriffsschicht darübergelegt – fehlt ein vorab spezifizierter, stabiler Integrationsvertrag, wie er in einem *API-first-Ansatz* üblich wäre. Dies erschwert insbesondere Integrationen mit Monitoring-Systemen, CI/CD-Pipelines oder externen Automatisierungsprozessen.

Das BFF schließt diese Lücke, indem es als Vermittlungsschicht zwischen Frontend, CMDB und weiteren Systemen fungiert. Es aggregiert und transformiert Daten aus verschiedenen Quellen, validiert Eingaben, setzt Sicherheits- und Zugriffskontrollen durch und stellt eine konsolidierte, integrations- und frontendgerechte API bereit. Zudem ermöglicht es Mechanismen wie Caching und die Entkopplung der Frontend-Kommunikation von internen Systemen. Dadurch kann sowohl Performance als auch Sicherheit erhöht und eine konsistente sowie erweiterbare Bereitstellung der CMDB-Daten gewährleistet werden.

Obwohl DataGerry bereits über eine eigene Benutzeroberfläche verfügt, wird im Architekturkonzept ein zusätzliches Frontend vorgesehen. Die vorhandene UI ist primär auf administrative Aufgaben und die generische Verwaltung von Konfigurationsobjekten ausgelegt, bietet jedoch nur eingeschränkte Möglichkeiten für organisationsspezifische Workflows, eigene Visualisierungen oder domänenspezifische Interaktionen. Durch ein dediziert entwickeltes Frontend können gezielte Nutzungsprozesse vereinfacht, komplexe Datenbeziehungen übersichtlich dargestellt und zusätzliche Funktionen implementiert werden, die über den Standardumfang von DataGerry hinausgehen. Das Frontend nutzt dabei exklusiv die abstrahierte API des BFF, wodurch eine klare Trennung zwischen Präsentation, Logik und Datenhaltung gewährleistet bleibt und die Anpassbarkeit des Systems langfristig sichergestellt wird.

## 4.3 Datenflüsse und Kommunikation

Alle Komponenten – einschließlich des Backend-for-Frontend (BFF) – kommunizieren über standardisierte REST-Schnittstellen und tauschen Daten ausschließlich über etablierte Protokolle (z. B. HTTPS, JSON) aus. Dies gewährleistet Interoperabilität[5] und erleichtert

[5]Der Begriff *Interoperabilität* beschreibt die Fähigkeit unterschiedlicher Systeme, Anwendungen oder Komponenten, miteinander zu kommunizieren, Daten auszutauschen und diese Informationen effizient weiterzuverarbeiten, ohne dass eine Anpassung der beteiligten Systeme erforderlich ist. Vgl. ISO/IEC 2382:2015, Information technology – Vocabulary.

zukünftige Integrationen.

- Der **Developer** interagiert über das Cockpit-Frontend mit der CMDB. Änderungen oder Abfragen werden über das Cockpit-BFF an die DataGerry-API weitergeleitet.
- Die **DataGerry-API** dient als zentrales Gateway zur Datenhaltung. Sie führt Schreib- und Leseoperationen auf der MongoDB aus.
- Das **Authentifizierungssystem (Keycloak)** validiert Benutzerzugriffe und gibt Access Tokens an die Kommunikationspartner zurück. Dabei wird OAuth2 eingesetzt. Eine vollständige OIDC-Kompatibilität besteht seitens DataGerry jedoch nicht, da lediglich der Token-basierte Zugriff unterstützt wird und keine vollständige Umsetzung der OIDC-Spezifikation (z. B. standardisierte Claims, UserInfo-Endpoint oder automatisiertes Rollenmapping) erfolgt.
- Über den **Datenmigrationsprozess** werden Informationen aus dem bestehenden Asset Inventory (Microsoft Access) in eine CSV exportiert und andschließend per CSV Import Funktion, welche die Datagerry UI bietet, in die CMDB überführt.

Die vollständige Darstellung der Datenflüsse, einschließlich der Netzwerkschichten und Clusterebenen, findet sich in Abbildung 1.

## 4.4 Integration und Sicherheit

Ein wesentliches Ziel des Architekturkonzepts besteht darin, die CMDB sicher in bestehende Unternehmenssysteme zu integrieren, ohne gegen vorhandene Sicherheitsrichtlinien zu verstoßen. Die einzelnen Komponenten werden zwar in Containern betrieben, jedoch stellt die Containerisierung allein keinen Sicherheitsgewinn dar. Sicherheit entsteht erst durch zusätzliche Maßnahmen wie Image-Hardening [6], regelmäßige Updates, kontrollierte Netzsegmentierung und das Prinzip der minimalen Berechtigungen.

Die Kommunikation zwischen den Modulen erfolgt über verschlüsselte HTTPS-Verbindungen, wobei die TLS-Terminierung typischerweise nicht in den einzelnen Containern, sondern an zentralen Infrastrukturkomponenten wie einem Reverse Proxy stattfindet. Dadurch können Zertifikate konsistent verwaltet und Verbindungen kontrolliert überwacht werden.

Für die Authentifizierung und Autorisierung kommt *Keycloak* zum Einsatz, das rollen- und gruppenbasierte Zugriffskontrollen bereitstellt. Sensible Anmeldeinformationen werden nicht in der CMDB selbst gespeichert. Der Einsatz von *CyberArk* kann hierbei sinnvoll sein, sofern er auf die Verwaltung kurzlebiger API-Tokens, Service-Accounts oder anderer betriebskritischer Zugangsdaten beschränkt bleibt. Eine direkte Kopplung zwischen

[6] Image-Hardening umfasst alle Maßnahmen zur Absicherung von Container- oder VM-Images, indem unnötige Komponenten entfernt, sichere Konfigurationen umgesetzt und Integritätskontrollen etabliert werden. Ziel ist es, die Angriffsfläche zu minimieren und die Images nahtlos in bestehende Sicherheits- und Integrationsprozesse einzubetten.

CMDB und *CyberArk* im Sinne eines automatischen Credential-Austauschs wäre hingegen sicherheitstechnisch kritisch und ist nicht vorgesehen.[20].

## 4.5 Bezug zum Feasibility Check

Der in Abbildung 2 dokumentierte *Feasibility Check* dient als Grundlage für die Bewertung der Umsetzbarkeit der vorgeschlagenen Architektur. Er überprüft die technische Eignung der eingesetzten Open-Source-Komponente *Datagerry*, hinsichtlich der Anforderungen die in Kapitel 3.2 und 3.3 erarbeitet wurden.

Gleichzeitig zeigt der Feasibility Check eine klare Trennung zwischen funktionalen und nicht-funktionalen Anforderungen: Während DataGerry die funktionalen Anforderungen – insbesondere hinsichtlich Flexibilität, Modellierbarkeit und Erweiterbarkeit der Konfigurationsobjekte – erfüllt, wird deutlich, dass die Lösung bei den nicht-funktionalen Anforderungen an ihre Grenzen stößt. Dies betrifft insbesondere Aspekte wie Verfügbarkeit, Performance bei großen Datenmengen, Skalierbarkeit sowie die Fähigkeit, definierte SLA-Vorgaben einzuhalten. DataGerry bietet keine native HA-Unterstützung, was für produktive CMDB-Systeme ein zentrales Kriterium darstellt. DataGerry selbst ist nicht HA-fähig. Die Verfügbarkeit muss über Infrastrukturkomponenten wie Container-Orchestrierung (z. B. Kubernetes), Load Balancer und MongoDB-Replica-Sets hergestellt werden. Diese Abhängigkeit von externer Infrastruktur bedeutet, dass hohe Verfügbarkeit, horizontale Skalierung oder Lastverteilung nicht durch die CMDB-Anwendung selbst, sondern ausschließlich durch zusätzliche Plattformkomponenten gewährleistet werden können. Insgesamt zeigt der Feasibility Check, dass DataGerry zwar als funktionale Grundlage und prototypische Implementierungsbasis geeignet ist, die Anforderungen an einen skalierbaren, SLA-konformen und hochverfügbaren Produktivbetrieb jedoch nicht vollständig erfüllt.[21].

## 4.6 Zusammenfassung

Das vorgestellte Architekturkonzept bietet eine skalierbare, modulare und sichere Grundlage für den Aufbau einer modernen CMDB. Durch den Einsatz von Open-Source-Technologien und standardisierten Schnittstellen wird die Interoperabilität mit bestehenden IT-Systemen gewährleistet. Der modulare Aufbau fördert dabei die langfristige Wartbarkeit und Anpassungsfähigkeit des Systems.

# 5 Integration und Automatisierung

## 5.1 Ziel und Bedeutung der Integration

Die Integration einer Configuration Management Database (CMDB) in bestehende IT-Prozesse bildet die Grundlage für eine effiziente, automatisierte und nachvollziehbare Infrastrukturverwaltung. Eine moderne CMDB wie *DataGerry* fungiert nicht als isoliertes System, sondern als zentrales Bindeglied zwischen Monitoring, Deployment und Change-Management-Prozessen. Im Sinne der DevOps-Philosophie wird dabei die enge Verzahnung zwischen Entwicklung, Betrieb und Automatisierung angestrebt [18].

## 5.2 Technologische Grundlagen

Zur Umsetzung der Integrations- und Automatisierungsstrategie werden Containerisierung, Infrastructure as Code (IaC)[7] sowie kontinuierliche Integrations- und Deployment-prozesse (CI/CD) genutzt. Die Containerisierung erfolgt mittels *Podman*, wodurch die CMDB und ihre Abhängigkeiten in standardisierte, portable Umgebungen verpackt werden können [22]. *Ansible* dient als Orchestrierungs- und Automatisierungs-Tool für die Bereitstellung der Systemkomponenten (*DataGerry*, *RabbitMQ*, *MongoDB*) [23]. Die Versionskontrolle und Integration in CI/CD-Pipelines wird über *Jenkins* sichergestellt.

## 5.3 Umsetzung der Automatisierung

### 5.3.1 Infrastructure as Code mit Ansible

Das Ansible-Playbook automatisiert die Bereitstellung der zentralen CMDB-Komponenten. Dabei werden folgende Hauptaufgaben realisiert:

- Installation und Konfiguration von *DataGerry*, *RabbitMQ* und *MongoDB*,
- Sicherstellung der Serviceabhängigkeiten (Startreihenfolge),
- Integration in das Netzwerkumfeld (Ports, Volumes, Security),
- Implementierung eines idempotenten Deployments, d. h. wiederholte Ausführungen führen zu keinem fehlerhaften Zustand.

Der modulare Aufbau des Playbooks erlaubt eine Wiederverwendung einzelner Rollen und Tasks für zukünftige Erweiterungen. Das Vorgehen folgt den Best Practices der IaC-Methodik, wonach Infrastrukturänderungen versioniert, dokumentiert und reproduzierbar

[7] Infrastructure as Code (IaC) bezeichnet den Ansatz, IT-Infrastruktur über Code zu definieren, zu verwalten und automatisiert bereitzustellen, anstatt sie manuell zu konfigurieren. Dadurch wird eine reproduzierbare, versionierbare und skalierbare Bereitstellung von Systemen ermöglicht. Vgl. Hüttermann, M. (2012): *DevOps for Developers*. Apress.

bereitgestellt werden sollen. Das im Listing 2 dokumentierte Ansible Playbook beschreibt folgenden Automatisierungsprozess: Lädt den DataGerry-Quellcode herunter, installiert notwendige Abhängigkeiten, kopiert Konfigurationsdateien (cmdb.conf, datagerry.conf) und richtet den Service (datagerry.service) ein [24].

### 5.3.2 Containerisierung mit Docker Compose

Die Containerisierung erfolgt über ein `docker-compose.yml`-File (vgl. Listing 1). Hier werden die Dienste `datagerry`, `rabbitmq` und `mongodb` als voneinander abhängige Container beschrieben. Docker Compose ermöglicht die einfache Verwaltung der Dienste über Befehle wie:

```
docker-compose up -d
docker-compose down
```

Diese Methode bietet Vorteile hinsichtlich Portabilität und Wiederholbarkeit, insbesondere bei Deployments auf Red Hat Enterprise Linux (RHEL 9) oder vergleichbaren Systemen [21].

### 5.3.3 Integration mit CI/CD

Die Integration der CMDB in bestehende CI/CD-Pipelines (Continuous Integration/Continuous Deployment) stellt einen wesentlichen Schritt dar, um Infrastruktur- und Anwendungsbereitstellungen automatisiert, konsistent und nachvollziehbar zu gestalten. Ziel ist es, Konfigurationsdaten aus der CMDB automatisiert in Build-, Test- und Deployment-Prozesse einzubinden, sodass Deployments stets auf aktuellen und verifizierten Systeminformationen basieren.

#### Automatisierte Bereitstellung von Konfigurationsdaten

CI/CD-Systeme wie Jenkins greifen nicht direkt auf die CMDB zu, sondern beziehen die benötigten Infrastruktur- und Servicedaten über das Backend-for-Frontend (BFF), das als vermittelnde Schicht dient. Das BFF ruft die Informationen über die REST-API von DataGerry ab, bereitet sie für Integrationsprozesse auf und stellt sie CI/CD-Pipelines in konsolidierter Form bereit. Typische Anwendungsfälle sind unter anderem:

- Dynamisches Abrufen von Server-, Cluster- oder Containerinformationen zur Laufzeit von Pipeline-Jobs,
- bereitstellung strukturierter Konfigurationsparameter – etwa IP-Adressen, Zielumgebungen oder Abhängigkeitsinformationen – die durch das BFF aus der CMDB abgefragt und CI/CD-Jobs als Variablen oder Artefakt-Dateien zur Verfügung gestellt werden. Die eigentliche Einbindung in Deployment-Skripte erfolgt anschließend durch die Pipeline selbst,

- Sicherstellung, dass Deployments nur auf Systeme durchgeführt werden, die in der CMDB als "bereit" oder "freigegeben" markiert sind.

## 5.4 Integration der CMDB mit Monitoring-Systemen

Zur Schaffung einer geschlossenen Systemlandschaft wird die CMDB über das *BFF* mit Monitoring-Systemen wie *Prometheus* integriert. Diese Systeme können automatisch Daten aus der CMDB abrufen, um beispielsweise neue Services oder Hosts zu überwachen. Durch diese Integration wird die Transparenz erhöht und eine automatische Synchronisierung zwischen Inventar und Überwachung erreicht (vgl. Use Case 2 in Kapitel 3.4). *DataGerry* bietet hierfür eine gut dokumentierte REST-API, die Abfragen, Filterungen und CRUD-Operationen auf Objekten erlaubt [1].

## 5.5 Zusammenfassung

Die prototypische Integration zeigt, dass sich durch den Einsatz von Automatisierungswerkzeugen wie Ansible und CI/CD-Pipelines einzelne Arbeitsschritte reproduzierbar und teilweise automatisiert ausführen lassen. Eine quantitative Bewertung der Verbesserungen – etwa in Form konkreter Messwerte zu Stabilität, Ausführungszeiten oder Fehlerreduktion – wurde im Rahmen dieser Arbeit jedoch nicht durchgeführt, sodass die Ergebnisse qualitativ zu interpretieren sind.

Die Umsetzung des Deployments mit Docker Compose ermöglicht eine vereinfachte Bereitstellung der benötigten Komponenten, stellt jedoch keine skalierbare oder hochverfügbare Umgebung bereit. Vielmehr dient Compose in diesem Kontext als leichtgewichtige Laufzeitumgebung für Entwicklungs- und Testzwecke. Für produktive Szenarien wären Orchestrierungswerkzeuge wie *Kubernetes* erforderlich, um Skalierung, Self-Healing oder Rolling Updates sicherzustellen.

Durch die Verwendung von Infrastructure-as-Code-Ansätzen werden Transparenz und Nachvollziehbarkeit der Konfiguration verbessert, dennoch handelt es sich nicht um eine vollständig automatisierte Umgebung. Verschiedene Schritte – beispielsweise Migrationsprozesse, Integrationslogiken oder sicherheitskritische Freigaben – erfordern weiterhin manuelle Entscheidungen. Die Architektur bleibt erweiterbar und bildet eine Grundlage, auf der zukünftige Integrationen, etwa mit ITSM- oder Backup-Lösungen, aufbauen können, ohne dass der Anspruch einer vollumfänglichen Automatisierung erhoben wird.

# 6 Fazit und Ausblick

## 6.1 Zusammenfassung der Ergebnisse

Die vorliegende Arbeit verfolgte das Ziel, ein detailliertes Infrastruktur- und Architekturkonzept für den Aufbau einer Configuration Management Database (CMDB) zu entwickeln. Im Mittelpunkt standen insbesondere moderne Open-Source-Technologien wie *DataGerry*, containerisierte Deployments auf Basis von *Docker*, Automatisierungsmethoden wie *Ansible* und *Infrastructure as Code (IaC)* sowie die Integration in bestehende Monitoring-Systeme und CI/CD-Pipelines.

Im Rahmen der Analyse wurden zunächst funktionale und nicht-funktionale Anforderungen sowie konkrete Use Cases erarbeitet, die als Grundlage für die konzeptionelle Ausgestaltung der CMDB-Architektur dienten. Auf dieser Basis erfolgte ein Feasibility Check, welcher die Eignung von *DataGerry* als Open-Source-Lösung für den Einsatz als zentrale CMDB evaluierte und dokumentierte. Das Ergebnis bestätigte die technische und funktionale Passfähigkeit der Plattform im Hinblick auf die ermittelten Anforderungen.

Anschließend wurde die konzeptionelle Architektur entwickelt. Hierzu entstand ein UML-Datenflussdiagramm, das die Interaktionen der zentralen Systemkomponenten abbildet. Die Architektur wurde in die Schichten *Präsentationsschicht*, *Applikationsschicht* und *Datenhaltungsschicht* unterteilt. Darüber hinaus folgt das Design einem modularen Aufbau, der eine flexible Erweiterbarkeit und Wartbarkeit des Gesamtsystems sicherstellt.Das vorgesehene Identity-Management basiert auf *Keycloak*, welches Rollen, Single-Sign-On und Zugriffskontrolle bereitstellt.

Zur praktischen Umsetzung wurde ein *Ansible*-Playbook implementiert, das die automatisierte Installation und Konfiguration der Kernkomponenten *DataGerry*, *RabbitMQ* und *MongoDB* ermöglicht. Diese Dienste wurden mithilfe einer *Docker-Compose*-Datei in voneinander abhängigen Containern orchestriert, wodurch eine konsistente und wiederholbare Bereitstellung gewährleistet wird.

Im weiteren Verlauf konnten erste *Configuration Items (CIs)* erfolgreich importiert werden. Hierzu wurde ein CSV-Export aus dem bestehenden Asset-Inventory-System erstellt und über die CSV-Import Funktion, welche vorab im Feasability überprüft wurde (vgl. Abbildung 2), auf der Administrationsoberfläche von *DataGerry* eingespielt (vgl. Abbildung 3).

Darüber hinaus wurde die geplante Integration der CMDB in die bestehende CI/CD-Pipeline auf Basis von *Jenkins* konzipiert. Diese Integration kann über die von *DataGerry* bereitgestellte REST-API realisiert werden, wodurch Konfigurationsdaten automatisiert in Build- und Deployment-Prozesse eingebunden werden können. Ein analoges Vorgehen ist für das Monitoring-System *Prometheus* vorgesehen, um Infrastrukturänderungen und Servicezustände dynamisch zu erfassen und auszuwerten.Zusammenfassend lässt sich festhalten, dass die Arbeit ein vollständiges Konzept für eine skalierbare, modular aufgebaute

und datenschutzkonforme CMDB liefert, die sich in moderne DevOps-Prozesse integrieren lässt.

## 6.2 Ausblick auf weiterführende Arbeiten

Obwohl das konzeptionelle Fundament gelegt ist, stehen für die praktische Umsetzung noch weitere Arbeitsschritte an. Erste Priorität besitzt die vollständige Implementierung und produktive Einführung der CMDB in die bestehende IT-Landschaft. Dabei müssen insbesondere folgende Punkte weiterentwickelt werden:

- **Automatisierte Discovery-Mechanismen:** Entwicklung von Skripten oder Agenten, die Infrastrukturkomponenten automatisch erkennen und in die CMDB einpflegen.
- **Change Management Workflow:** Integration mit ITSM-Tools (z. B. ServiceNow) zur automatisierten Erfassung von Changes und deren Auswirkungen.
- **Monitoring- und Alarmierungskonzepte:** Automatische Ableitung von Überwachungskonfigurationen aus der CMDB.
- **Security & Compliance:** Umsetzung von Privacy by Design, Zugriffsrichtlinien und DSGVO-Anforderungen im laufenden Betrieb.
- **Test- und Abnahmeprozesse:** Definition von Testplänen, Continuous Deployment Pipelines und Abnahmekriterien für neue CMDB-Funktionen.
- **Dokumentation und Schulung:** Erstellung von Nutzerhandbüchern sowie Schulung von Administratoren und Endanwendern.

Langfristig kann die CMDB als Grundlage für weiterführende Technologien wie Machine Learning (z. B. Predictive Maintenance), automatisierte Impact-Analysen oder Self-Healing-Infrastrukturen dienen. Ebenso bietet sich die Möglichkeit, Knowledge Graphs oder Semantik-Technologien zur intelligenten Datenverknüpfung einzusetzen.

Die erarbeiteten Konzepte bilden somit nicht den Endpunkt, sondern vielmehr den Startschuss für die technische Realisierung einer zukunftsfähigen, transparenten und resilienten IT-Infrastruktur.

# 7 Anhang

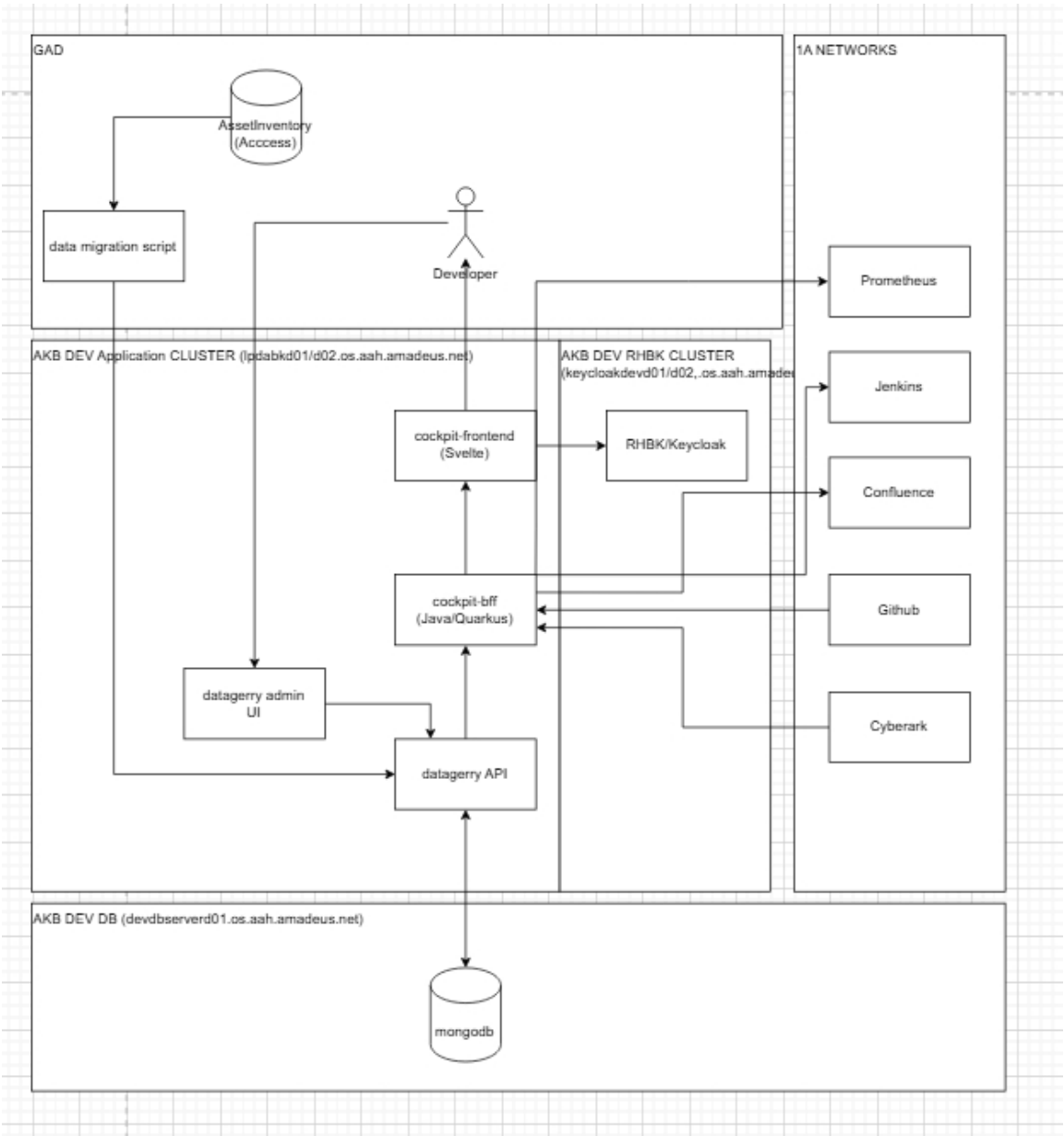



Abbildung 1: Datenflussdiagramm des CMDB-Projekts.

Feasibility Check – DataGerry

1. Grundfunktionen – Datenmodell & CRUD
 Typen erstellen, Felder definieren, Daten einfügen, lesen, updaten, löschen – vollständig über UI & API möglich.

2. Relationen & Referenzen
 1:n und n:m Beziehungen, Objektverknüpfungen, verschachtelte Relationen – solide umgesetzt.

3. Filter & Abfragen
 Volltextsuche, Attribut- und Relationsfilter, Sortierung, Pagination – vorhanden.

4. Import / Export
 CSV/JSON Import & Export, Backups über MongoDB, kein direkter DB-Import.

5. API-Zugriff
 REST API mit Token-Auth, CRUD & Filterung, Swagger-Docs verfügbar.

6. Benutzer, Rollen & Sicherheit
 Benutzerverwaltung, ACL, LDAP/SSO Unterstützung, Rechte auf Typ-/Feldebene.

7. Benutzeroberfläche & Usability
 Modernes Web-UI, Audit-Logs, visuelle Differenzierung von Typen & Objekten.

8. Erweiterbarkeit & Wartbarkeit
 Templates, MDS, flexible Felder, Backups, dokumentierte Architektur.

9. Sonderfunktionen
 Versionierung, Bulk-Änderungen, Reporting, Webhooks, CI Explorer.

Fazit: DataGerry erfüllt alle Kernanforderungen einer CMDB und ist für produktive Nutzung geeignet.

Abbildung 2: Zusamenfassung des Feasability Check - Datagerry.

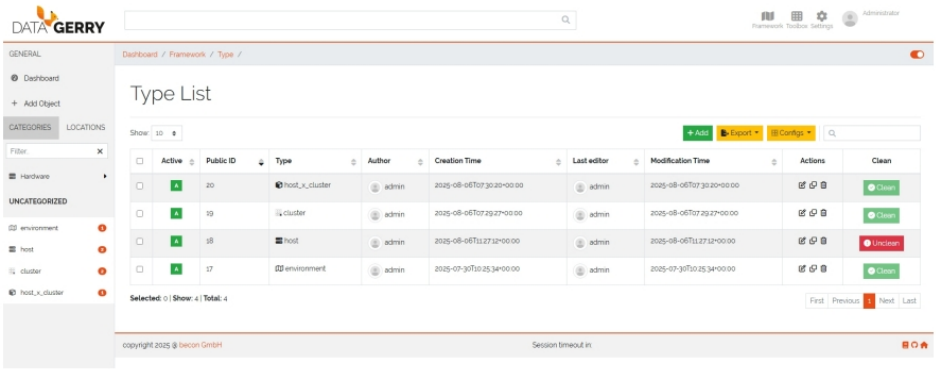



Abbildung 3: Weboberfläche von Datagerry

```yaml
version: "3.0"
services:
  nginx:
    image: docker.io/becongmbh/nginx:latest
    hostname: nginx
    ports:
      - "80:80"
      - "443:443"
    depends_on:
      - datagerry
    environment:
      NGINX_SSL_CERT: "/data/cert/cert.pem"
      NGINX_SSL_KEY: "/data/cert/key.pem"
      NGINX_LOCATION_DEFAULT: "/;/;http://datagerry:4000"
    restart: unless-stopped
    volumes:
      - ./cert:/data/cert

  datagerry:
    image: docker.io/datagerry/datagerry:latest
    hostname: datagerry
    depends_on:
      - db
      - broker
    environment:
      DATAGERRY_Database_host: "db"
      DATAGERRY_MessageQueueing_host: "broker"
    restart: unless-stopped

  db:
    image: docker.io/mongo:4.4.29
    hostname: db
    restart: unless-stopped
    volumes:
      - mongodb-data:/data/db
      - mongodb-config:/data/configdb

  broker:
    image: docker.io/rabbitmq:3.8
    hostname: broker
    restart: unless-stopped
    volumes:
      - rabbitmq-data:/var/lib/rabbitmq

volumes:
  rabbitmq-data:
```

```
  mongodb-data:
  mongodb-config:
```

Listing 1: Docker-Compose.yml Datei zur Bereitstellung von DataGerry RabbitMQ und MongoDB

```
- name: Git & Python vorbereiten
  apt:
    name:
      - git
      - python3
      - python3-venv
    state: present

- name: Datagerry klonen
  git:
    repo: "{{ datagerry_repo_url }}"
    dest: "{{ datagerry_install_path }}"
    version: "{{ datagerry_repo_branch }}"

- name: Virtuelle Umgebung fuer DataGerry erstellen
  command: python3 -m venv /opt/datagerry/venv
  args:
    creates: /opt/datagerry/venv

- name: Pip in der virtuellen Umgebung aktualisieren
  command: /opt/datagerry/venv/bin/pip install --upgrade pip

- name: Python-Abhaengigkeiten in der virtuellen Umgebung installieren
  command: /opt/datagerry/venv/bin/pip install -r /opt/datagerry/
     requirements.txt

- name: Konfigurationsdatei bereitstellen
  template:
    src: datagerry.conf.j2
    dest: /opt/datagerry/datagerry.conf
    owner: root
    group: root
    mode: '0644'

- name: Datagerry starten (einmalig)
  shell: |
    nohup /opt/datagerry/venv/bin/python /opt/datagerry/datagerry.py --
       config /opt/datagerry/datagerry.conf &
  args:
    chdir: /opt/datagerry
    creates: /tmp/datagerry_started
```

Listing 2: roles/datagerry/tasks/main.yml Ansible Playbook zur Installation von Datagerry

# Literatur


[1] becon GmbH, "Datagerry documentation," 2024. Offizielle Dokumentation der Open-Source-CMDB DataGerry.

[2] I. G. I. . U. of Göttingen, "State of the nation survey findings — cms/cmdb." https://www.uni-goettingen.de/de/document/download/d895466597b111a39f3592cac5e93ddb-en.pdf/itil_state_of_the_nation_survey.pdf, 2025. abgerufen am: 5.12.2025.

[3] AXELOS, *ITIL Foundation: ITIL 4 Edition*. London: The Stationery Office (TSO), 2019.

[4] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[5] G. Inc., "Understanding and avoiding vendor lock-in in cloud computing," 2023.

[6] I. Sommerville, *Software Engineering*. Harlow: Pearson Education Limited, 10th edition ed., 2016.

[7] K. Wiegers and J. Beatty, *Software Requirements*. Redmond, WA: Microsoft Press, 4th edition ed., 2023.

[8] IEEE, "Ieee 830-1998: Recommended practice for software requirements specifications," 1998. Institute of Electrical and Electronics Engineers.

[9] O. of Government Commerce, *ITIL Service Transition*. London: The Stationery Office (TSO), 2007.

[10] R. S. Pressman and B. R. Maxim, *Software Engineering: A Practitioner's Approach*. McGraw-Hill Education, 9 ed., 2020.

[11] Red Hat, Inc., *Red Hat Enterprise Linux 9 System Design Guide*. 2024.

[12] P. A. Laplante, *Requirements Engineering for Software and Systems*. CRC Press, 3 ed., 2017.

[13] N. I. of Standards and T. (NIST), "Security and privacy controls for information systems and organizations (nist sp 800-53, rev. 5)," tech. rep., U.S. Department of Commerce, 2020.

[14] "Verordnung (eu) 2016/679 des europäischen parlaments und des rates vom 27. april 2016 zum schutz natürlicher personen bei der verarbeitung personenbezogener

daten (datenschutz-grundverordnung, dsgvo)." https://eur-lex.europa.eu/eli/reg/2016/679/oj, 2016. Amtsblatt der Europäischen Union, L 119, 4.5.2016.

[15] B. Burns, *Designing Distributed Systems: Patterns and Paradigms for Scalable, Reliable Services*. O'Reilly Media, 2016.

[16] MongoDB Inc., *MongoDB Documentation, Version 7.0*, 2023.

[17] "It infrastructure management best practices." https://www.gartner.com/en/documents, 2022.

[18] G. Kim, J. Humble, P. Debois, and J. Willis, *The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations*. IT Revolution Press, 2017.

[19] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*. Boston: Addison-Wesley, 4th ed., 2021.

[20] W. Stallings, *Network Security Essentials: Applications and Standards*. Boston: Pearson, 7th ed., 2021.

[21] D. Merkel, "Docker: Lightweight linux containers for consistent development and deployment," *Linux Journal*, vol. 239, 2014.

[22] Docker Inc., "Docker documentation." https://docs.docker.com/, 2025.

[23] Red Hat Inc., "Ansible documentation." https://docs.ansible.com/, 2025.

[24] M. Hüttermann, *DevOps for Developers*. Apress, 2012.