Fachhochschule Nordwestschweiz
Hochschule für Informatik

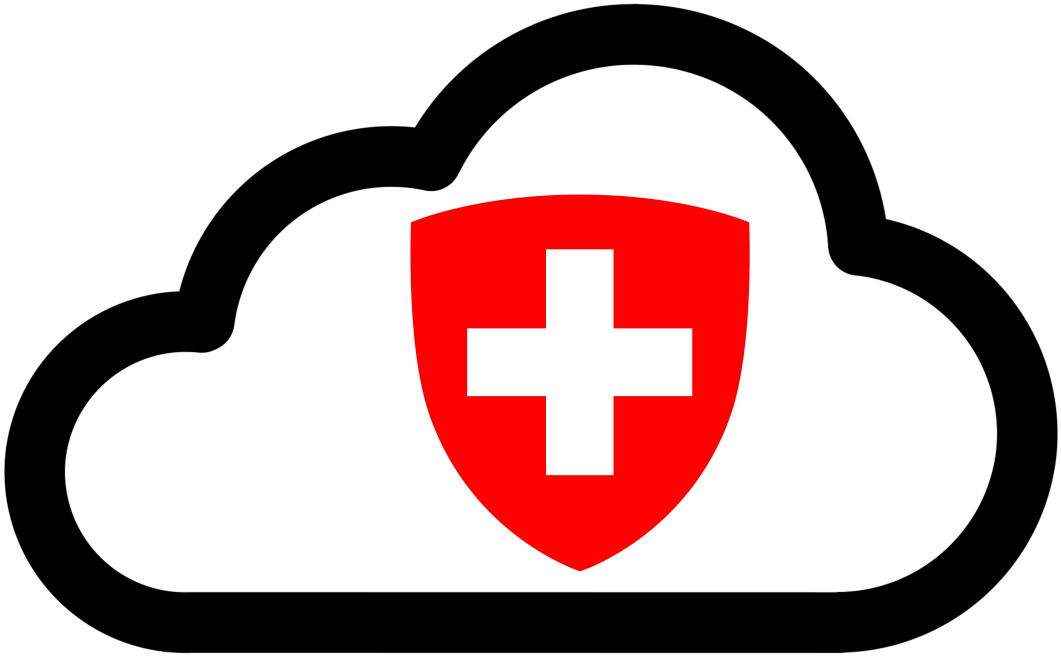
Schweizerische Eidgenossenschaft
Confédération suisse
Confederazione Svizzera
Confederaziun svizra

Eidgenössisches Finanzdepartement EFD
Bundesamt für Informatik
und Telekommunikation BIT

Enterprise-Architektur von multicloud-konformen Landingzones

Bachelorthesis

Windisch, März 2026



Studentin/Student	Frithjof Hoppe Benjamin Dätwyler
Expertin/Experte	Romano Roth
Fachbetreuer/in	Sebastian Graf
Auftraggeber	Sebastian Matyas
	Thierry Perroud
Projektnummer	25HS_IMVS17

Fachhochschule Nordwestschweiz, Hochschule für Informatik

Abstract

Organisationen mit Hybrid-Multi-Cloud-Strategien stehen vor der Herausforderung, unterschiedliche plattformspezifische Strukturmodelle konsistent zu steuern. Für das Bundesamt für Informatik und Telekommunikation (BIT) fehlte ein Provider-agnostischer architektonischer Leitrahmen zur Gestaltung einer Landing Zone (LZ). Ziel dieser Arbeit war die Entwicklung eines entsprechenden Architekturansatzes auf Enterprise-Architecture (EA)-Ebene.

Hierzu wurde ein reduziertes EA*-Framework angewendet, um mittels Capability-Map und Use-Cases einen Anwendungskontext abzuleiten. Darauf aufbauend wurde eine LZ*-Architektur entwickelt und exemplarisch durch eine Traceability-Architektur in einem Proof of Concept konkretisiert. Die Architekturen wurden anhand von Architekturprinzipien, Qualitätsattributen und Trade-off-Kriterien evaluiert.

Die Ergebnisse zeigen, dass eine Provider-agnostische Strukturierung von einer LZ* möglich ist und einen konsistenten Governance-Leitrahmen für Hybrid-Multi-Cloud-Umgebungen unterstützen kann.

Keywords:

Landing Zone, Hybrid Multi Cloud, Enterprise Architecture, Governance, Traceability, Architekturprinzipien

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Akronyme

ADM Architecture Development Method. v, vii, 5–7, 21, 99

BIT Bundesamt für Informatik und Telekommunikation. ii, v, vii, 1, 2, 6–8, 14, 15, 18, 21, 22, 29, 31, 36, 40, 44, 45, 81, 92–94, 97
BV Bundesverwaltung. 1, 3, 62

CSB Cloud Service Broker. 1

EA Enterprise-Architecture. ii, vii, 1, 2, 5, 6, 22–25, 44–46, 81, 94, 96–98

IaC Infrastructure as Code. 45, 49, 56, 60, 62, 90
IAM Identity and Accessmanagement. 3, 40, 44

LZ Landing Zone. ii, v–vii, 1–3, 5, 7–10, 13–15, 18, 38–41, 44–46, 48–52, 54–56, 58, 61, 62, 66, 67, 70, 72, 81, 87–89, 93–99

MCDA Multiple-criteria decision analysis. 32
MVP Minimum Viable Product. 37

NFR Non-Functional Requirements. 15, 21, 99
NIST National Institute of Standards and Technology. 3, 81, 94

PoC Proof of concept. v–vii, 2, 7, 37, 38, 40–43, 46, 62, 63, 66–68, 70–74, 78, 92–94, 96, 99, 100

Si001 Si001 - IKT Grundschutz der Bundesverwaltung. 3, 32, 42, 45, 51, 62, 81, 84, 86, 87, 94

TOGAF The Open Group Architecture Framework. vii, 5–7, 14, 21, 94

UC Use-Case. 34

1 Einleitung

Die Ausgangslage ist, dass die Bundesverwaltung (BV) eine Hybrid-Multi-Cloud-Strategie verfolgt, welche die Nutzung unterschiedlicher Cloud-Anbieter sowie eigener Infrastrukturen kombiniert. Das Bundesamt für Informatik und Telekommunikation (BIT) nimmt dabei die Rolle eines Cloud Service Broker (CSB) wahr. Das BIT* stellt die Anbindung an verschiedene Hyperscaler sicher und betreibt gleichzeitig eine eigene private Cloud-Infrastruktur on-premise. Die konkrete Ausführung der Strategie wird im Rahmen einer öffentlichen Ausschreibung ermittelt. Vor deren Abschluss existieren keine Präferenzen bezüglich Cloud-Anbietern, sowohl für den Public-Cloud- als auch den Private-Cloud-Aspekt der Strategie.

Eine Nutzung der verschiedenen Cloud-Plattformen setzt voraus, dass eine grundlegende Struktur existiert, auf der die Workloads der Anwender innerhalb der BV* aufbauen können. Die Plattform-Provider unterscheiden sich in der Art und Weise, wie ihre Plattform strukturiert ist (Hoppe & Dätwyler, 2025). Ideen oder Konzepte werden zwar häufig zwischen den Providern geteilt, die Ausgestaltung unterscheidet sich jedoch.

In der praktischen Umsetzung bedeutet dies für eine Organisation wie das BIT*, dass die einzelnen Cloud-Plattformen weitgehend isoliert betrachtet und betrieben werden müssen. Sicherheits- und Governance-Mechanismen werden plattformspezifisch implementiert, wodurch Inkonsistenzen entstehen können.

Gleichzeitig sinkt die Wiederverwendbarkeit von Architektur- und Betriebsmodellen über Plattformgrenzen hinweg. Dies erschwert eine einheitliche Durchsetzung von Governance-Vorgaben sowie die vergleichbare Steuerung von Workloads in einer Hybrid-Multi-Cloud-Umgebung. Ohne eine übergreifende Struktur droht eine Fragmentierung der Cloud-Architekturen innerhalb der Organisation.

Das BIT* strebt deshalb eine Homogenisierung mit dem Konzept einer Landing Zone (LZ) an. Diese LZ* wird vom BIT* als eine durch die Vorgaben der Plattform-Provider vordefinierte Struktur verstanden, in welcher die effektiven Angebote bezogen werden können bzw. Workloads der Anwender laufen. Dieses Verständnis der LZ* dient als Arbeitshypothese. Die Idee der LZ* soll trotz unterschiedlicher Cloud-Plattformen einheitlich angewendet werden können. Eine Vereinheitlichung würde es dem BIT* ermöglichen, einen Leitrahmen zu definieren, welcher für Projekte auf verschiedenen Cloud-Plattformen gilt.

Ziel dieser Arbeit ist die Entwicklung eines Provider-agnostischen Architekturansatzes für LZ* in Hybrid-Multi-Cloud-Umgebungen. Der Ansatz soll es ermöglichen, zentrale LZ*-Komponenten systematisch zu identifizieren und in einer konsistenten technischen Gesamtarchitektur zu strukturieren. Die Eignung dieses Ansatzes wird exemplarisch anhand einer Teilarchitektur untersucht und anhand definierter Evaluationskriterien bewertet.

Mit der aufgezeigten Problemstellung und Zielsetzung werden folgende Fragestellungen formuliert, welche im Rahmen dieser Arbeit beantwortet werden:

- *Welche Komponenten machen eine Landingzone für eine Hybrid-Multi-Cloud-Umgebung aus?*
- *Wie sieht eine geeignete technische Architektur einer Landingzone für eine Hybrid-Multi-Cloud-Umgebung aus und was muss sie erfüllen?*

Zur Beantwortung der Forschungsfragen wird in Abschnitt 3 das Problem aus Sicht des BIT* im Rahmen eines reduzierten Enterprise-Architecture (EA)-Ansatzes methodisch strukturiert. Auf dieser Grundlage wird mittels einer Capability-Map sowie ausgewählter Use-Cases der Anwendungskontext für einen LZ*-Architekturansatz hergeleitet.

Darauf aufbauend wird in Abschnitt 4 eine elementare LZ*-Architektur auf EA*-Ebene entwickelt und in Abschnitt 5 im Rahmen eines Proof of concept (PoC) exemplarisch anhand einer Traceability-Architektur konkretisiert.

Die entwickelten Architekturen werden anschliessend anhand definierter Architekturprinzipien, Qualitätsattribute sowie Trade-off-Kriterien evaluiert. Auf Basis dieser Evaluation werden in Unterabschnitt 7.2 Empfehlungen für den spezifischen Kontext des BIT* abgeleitet. Abschliessend werden in Abschnitt 8 die Ergebnisse im Hinblick auf die Forschungsfragen zusammengeführt und eingeordnet.

1.1 Erkenntnis

Die Recherche der Grundlagen macht ersichtlich, dass kein einheitliches Konzept für eine LZ* existiert. Basierend auf dem gewählten Ansatz wird deshalb ein Vorschlag für eine einheitliche LZ*-Architektur auf EA*-Ebene gemacht. Die Arbeit zeigt, dass sich eine multi-cloud-fähige LZ*-Architektur mithilfe von Architekturprinzipien und einer technischen Capability-Map systematisch strukturieren lässt. Der gewählte Ansatz erlaubt eine Business-getriebene Betrachtung der Architektur und unterstützt die Zerlegung in Teilarchitekturen. Exemplarisch wird dabei eine Traceability-Architektur untersucht.

2 Grundlagen und Stand der Technik

Die Nutzung des Begriffs LZ* beschränkt sich in der theoretischen Literatur mehrheitlich auf Referenzen durch Cloud-Plattformen selbst und wird nicht explizit aufgegriffen. Ebenso finden sich keine ausführlichen Konzepte dazu, wie LZ* zu strukturieren sind. In der Praxis der Cloud-Plattformen hingegen existiert ein genaueres Verständnis davon, worum es sich bei einer LZ* handelt. So wird sie nach Microsoft, 2025 bspw. als Möglichkeit zum Einrichten/Verwalten der Kundenumgebung bezeichnet, mit dem Ziel, Konsistenz in Bezug auf Sicherheit, Compliance und Betriebseffizienz zu gewährleisten. Dieses Ziel kann als einer der treibenden Faktoren betrachtet werden, warum LZ* relevant sind. Zusammenfassend kann festgehalten werden, dass durch eine LZ* die Cloud-Plattformen grundsätzlich verfügbar und vor allem auf sichere Art für eine Organisation nutzbar gemacht werden (Mulder, 2020a, Kapitel 6). Eine sichere Nutzung bedingt zunächst, dass die Verantwortlichkeiten zwischen Cloud-Plattform und Nutzer klar sind. Mit dem Modell der geteilten Verantwortung (oder auch Shared Responsibility Model) zeigen Plattformen deshalb auf, welcher Teil der Sicherung durch den Kunden oder durch die Cloud-Plattform selbst erbracht werden muss (Security, n. d.). Für Organisationen heisst dies häufig, dass sie sich bewusst werden müssen, welche Anforderungen sie selbst haben. Diese können bspw. sowohl regulatorisch aus Dokumenten wie Si001 - IKT Grundschutz der Bundesverwaltung (Si001) als auch organisatorisch durch Expertenteams oder Plattformteams begründet sein. Die relevanten Themen ähneln sich dabei über die Cloud-Plattformen hinweg. Identity and Accessmanagement (IAM), Network und Security sind dabei nur einige Beispiele.

Trotz der Gemeinsamkeit, dass Governance als notwendig verstanden wird, weisen die Cloud-Plattformen jedoch auch Unterschiede auf. Diese erstrecken sich von eigenen Metamodellen bis hin zu unterschiedlichen Implementationen von Komponenten. Eine Organisation, welche eine sichere Nutzung im Multi-Cloud-Kontext sicherstellen möchte, steht somit vor der Herausforderung, wie sie die verschiedenen Cloud-Plattformen verwalten soll. Hierfür existieren auf dem Markt bereits diverse Cloud-Management-Plattformen (bspw. Flexera One, Cloud Bolt, HPE Morpheus), welche sich hauptsächlich auf die operative Verwaltung und Bereitstellung von Cloud-Ressourcen spezialisieren. Anders ausgedrückt wird adressiert, wie die konkrete Implementation in einem Multi-Cloud-Setup möglich ist. Offener dagegen ist, wie die Strukturierung der Cloud-Ressourcen aussieht und welche Policies aus welchen Gründen einer Cloud-Ressource zugewiesen werden müssen. Cloud-Plattformen versuchen diese Governance zwar individuell abzubilden, wodurch jedoch ein übergreifender Ansatz fehlt. Diese Arbeit fokussiert sich deshalb auf einen Abstraktionsansatz für Governance und nicht auf die Entwicklung einer weiteren Cloud-Management-Plattform.

Für einen Abstraktionsansatz sind hierbei konkret die Security Frameworks relevant, deren Compliance sichergestellt werden soll. Ein prominentes Beispiel für die BV* ist Si001*, welches den IKT-Grundschutz der BV* beschreibt. Ein anderes bekanntes Framework ist die National Institute of Standards and Technology (NIST) Special Publication 800-53. Beide genannten Regulationen weisen mit Controls eine im Kern gleiche Struktur auf. Wichtiger jedoch ist, dass die korrekte Anwendung der Regulationen für Organisationen, welche sie nutzen, indirekt ein Ziel darstellt. Die Motivation kann dabei unterschiedlich sein (gesetzliche Vorgaben, selbst auferlegt etc.).

Anhand dieser Regulationen stellt sich die Frage, auf welcher Ebene eine solche Governance-Strukturierung erfolgen kann. Da Anforderungen aus regulatorischen Vorgaben oder organisationalen Zielsetzungen letztlich in technische Architekturentscheidungen überführt werden müssen, kann die Gestaltung von LZ* als ein architektonisches Problem verstanden werden.

Insbesondere im Multi-Cloud-Kontext entsteht dadurch die Notwendigkeit, eine Abstraktionsebene zu finden, welche unabhängig von spezifischen Implementationen einzelner Cloud-Provider

funktioniert. Eine mögliche Perspektive hierfür bietet die Betrachtung von Fähigkeiten (*Capabilities*) einer Organisation. Capabilities beschreiben relativ stabile, Technologie-unabhängige Möglichkeiten zur Erbringung bestimmter Leistungen, wie etwa das Management von Identitäten, die Segmentierung von Netzwerken oder die Überwachung von Systemzuständen.

Im Kontext von Enterprise-Architecture-Frameworks dienen Capabilities häufig als strukturierendes Element, um strategische Anforderungen mit konkreten Architekturartefakten in Beziehung zu setzen. Dadurch entsteht eine Grundlage, um Governance-Mechanismen wie Policies oder Controls systematisch einzuordnen und deren Umsetzung nachvollziehbar zu gestalten.

3 Methodik und Vorgehen

In diesem Abschnitt wird die Struktur aufgezeigt, welche letztendlich zu den vorgeschlagenen Konzepten in Abschnitt 4 (Landing Zone Architektur) führt. Zu diesem Zweck wird in Unterabschnitt 3.1 (Forschungsmethodischer Ansatz) zuerst die allgemeine Vorgehensweise aufgezeigt, deren Anwendung in den weiteren Unterabschnitten dargestellt wird.

3.1 Forschungsmethodischer Ansatz

Das Konzept der LZ* wird bei verschiedenen Cloud-Plattformen innerhalb der Cloud-Migrationsstrategien beschrieben. Die Treiber für die grundlegenden Entscheidungen scheinen in Migrationsvorhaben jeweils die Bedürfnisse jeweils die Bedürfnisse der Organisation zu sein Orban, 2017. Daraus wird die Hypothese abgeleitet, dass die LZ* selbst sowie deren verschiedene Komponenten dieser auf letztlich auf Business-Needs zurückzuführen sind. Diese Hypothese lässt sich mit verschiedener Fachliteratur in den Themen Multi-Cloud-Strategie sowie Multi-Cloud-Architektur untermauern AG, 2024, Mulder, 2020b, McCabe, 2024. Mulder, 2020b sieht konkret die Übersetzung von Business-Requirements zu einer multi-Cloud Strategie und schlussendlich zu der Strukturierung einer LZ* vor. Diese wird hierbei in das Gerüst einer EA* eingebettet.

Mulder, 2020b führt aus, dass für eine EA* bereits verschiedene Frameworks existieren in welche der Multi-Cloud-Kontext Kontext eingebettet werden kann, weshalb The Open Group Architecture Framework (TOGAF) als geeigneter etablierter Vertreter vorgeschlagen wird. TOGAF* sieht für die Erstellung einer EA* einen iterativen Zyklus namens Architecture Development Method (ADM) vor. Dies geschieht vor dem Hintergrund, dass sich die Grundlage für eine EA* durch ändernde Anforderungen aus dem Business ebenfalls ändert und entsprechend anpassen muss.

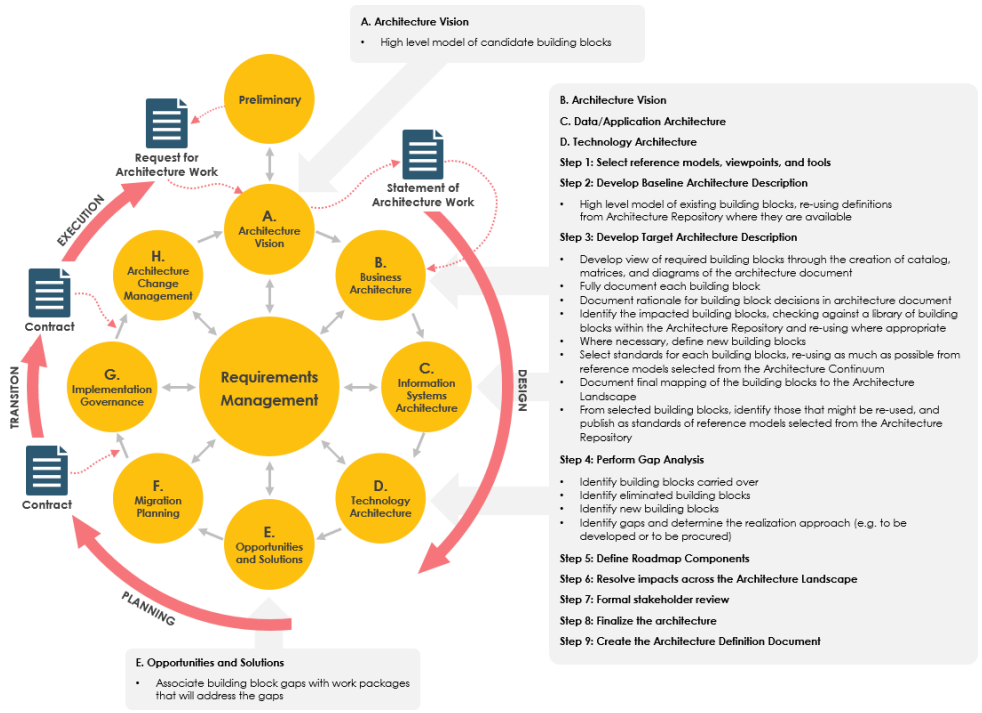


Abbildung 3.1: Kompletter ADM* Cycle Paradigm, 2025

Der in der Arbeit erstellte forschungsmethodische Ansatz lehnt sich gedanklich an der Methodik von Mulder, 2020b und TOGAF* an. Dies bedeutet, dass die Schritte in der erstellten Methodik

auf Phasen in TOGAF* zurückführbar sind (siehe Tabelle 3.2). Die Motivation dahinter ist, dass ein auf Business-Needs basierender Ansatz gewählt wird. Dadurch soll vermieden werden, dass der durch die Methodik erarbeitetet Outcome losgelöst vom Bedarf des Auftraggebers / des BIT* und potenziell weiteren Organisationen ist. Die EA* dient somit nur als Gerüst und es ist nicht angedacht eine komplette EA* zu definieren. Im Folgenden ist der angepasste Ansatz beschrieben, welcher verfolgt wird. Dieser zeigt welche Elemente mit welcher Begründung in der jeweiligen Phase des ADM* Cycle zur Anwendung kommen.

Phase TOGAF*	**Anwendung für Arbeit**	**Begründung**
Preliminary	Übernahme von existierenden Architekturprinzipien vom BIT* mit Ergänzung von etwaigen solchen für den Multi-Cloud-Kontext. Definieren eines fachspezifischen Metamodells mit LZ* als zentralen Punkt. Recherche der LZ* Komponenten, die bereits bei Anbietern existieren	Die Entwicklung des Outcomes soll sich gedanklich bereits an den Prinzipien des BIT* orientieren. Deshalb ist eine Verwendung von etablierten Architekturprinzipien essenziell. Das Metamodell erlaubt es wiederum ein gemeinsames Verständnis zu schaffen welche Artefakte geschaffen werden sollen und wie Elemente welche out-of-scope sind mit dem Kontext der Arbeit zusammenhängen.
A Architecture Vision	Definieren, welche Akteure an dem Outcome beteiligt sind. Ableiten von Scopes und Zielen basierend auf Annahmen. Definieren von elementaren / Level 1 Technical Capabilities	Die Capability-Map erlaubt es die Annahmen zu visualisieren welche Fähigkeiten das BIT mit dem Multi-Cloud-Ansatz erreichen möchte.
B Business Architecture	Definieren von granulareren Level 2 Capabilities und Use-Cases basierend auf der Priorisierung der L2 Capabilities durch die Stakeholder. Entwerfen von verschiedenen Architektur-Varianten für eine der priorisierten Capabilities	Durch die Granularität wird eine bessere Bewert- und Analysierbarkeit ermöglicht. Die Use-Cases erlauben anschliessend den Scope zu setzen in welcher die gewählte Architektur-Variante definiert wird.
C Information System Architectures		
D Technology Architecture		
E Opportunities and Solutions	Implementation eines PoC* basierend auf der gewählten Architektur-Variante	Der PoC* ermöglicht die Evaluierung der Machbarkeit des gewählten Ansatzes und dient somit als Validierungswerkzeug
F Migration Planning		
G Implementation Governance	Auswahl einer Architektur, welche den Evaluationskriterien gerecht wird, welche unter anderem auf dem Use-Case des BIT* basieren. Definieren der Artefakte welche aus der Wahl für die Variante hervorgehen (bspw. Anforderungen an die LZ*, Governance-Regeln, etc.)	Der Outcome der Evaluation bildet unter anderem die Antwort auf die initialen Forschungsfragen ab und ist somit essenziell
H Architecture Change Management	-	Nicht anwendbar, da mit dem Output des ADM*-Cycles kein produktiver Zustand, eine Übergabe an den Betrieb oder Weiterentwicklung angestrebt wird

Tabelle 3.2: Forschungsmethodik gegliedert nach den Phasen von ADM* in TOGAF*

3.2 Recherche von Landing Zone/-Komponenten bei Cloud-Plattformen

Zum Verständnis, wie der LZ*-Begriff, LZ* und LZ*-Komponenten bereits bei Cloud-Plattformen verbreitet sind, wird eine Recherche gemacht. Das Vorgehen der Recherche besteht aus den folgenden Schritten:

1. Recherche zur Etablierung des Begriffs LZ* in der wissenschaftlichen Literatur
2. Unterschiede in der Beschreibung der LZ*s als Begriff und die technischen Komponenten der LZ* bei relevanten Anbietern identifizieren
3. Abgrenzung der LZ*s zu anderen Ressourcen auf einer Cloud-Plattform

Als Begrenzung für die Recherche sollen lediglich relevante Cloud-Provider betrachtet werden. Amazon mit der Plattform AWS, Google mit GCP und Microsoft mit Azure bilden nach Saraswat und Tripathi, 2020 die drei Markt-stärksten Anbieter von umfassenden Cloud-Services. Zu den genannten Anbietern werden zusätzlich noch IBM und Alibaba wegen Verträgen im Rahmen des Vorhabens «Public Clouds Bund» der WTO-20007 nach BK, 2025 ebenfalls einbezogen, da diese vom BIT* verwendet werden. Diese Anbieter werden im Folgenden weiter als Cloud-Provider verstanden.

In einer initialen systematischen Literaturrecherche wird zunächst ermittelt, wie etabliert der Begriff LZ* bereits in wissenschaftlichen Arbeiten und der Industrie ist. Hierzu soll die Arbeitshypothese darüber, was eine LZ* ist (siehe Problemstellung), welche vom BIT* gefasst wurde, hinterfragt werden. Zur Recherche wird hierfür zu Beginn mit einer allgemeinen Recherchefrage gestartet, welche anschliessend basierend auf der in den Ergebnissen verwendeten Terminologie reformuliert wird. Die initiale Frage ist «What are the key architectural components and design principles for establishing a secure and efficient cloud landing zone?». Diese Recherchefrage wird anschliessend an ein KI-basiertes Tool übergeben, welches mittels semantischer Ähnlichkeit von Begriffen nach existierenden wissenschaftlichen Publikationen sucht (siehe Tools in Abschnitt 9 (Hilfsmittel)). Die angebundenen Engines sind hierbei Semantic Scholar und OpenAlex („Elicit's source for papers", 2025), welche etabliert sind. Der dadurch generierte Report zeigt Elicit, 2025 (siehe Appendix C), dass bis auf eine Quelle die Terminologie LZ* nicht explizit verwendet wird. Vielmehr werden auf grundsätzliche architektonische Konzepte hingewiesen, welche mit der Idee der LZ* zusammenhängen. Quellen, welche den Begriff verwenden, sind bis auf wenige Ausnahmen (Laheri, 2025) Dokumente der Cloud-Provider selbst oder Fachliteratur, welche indirekt auf die konkrete Implementierung von Cloud-Providern verweist, wie bei Mulder, 2020a. Eine Suche mit ähnlichen Schlüsselwörtern auf einer traditionellen Plattform [1] stützt diese Vermutung.

Dies wirft somit die Frage auf, ob es sich möglicherweise um einen Begriff handelt, der von Cloud-Providern selbst etabliert wurde und nicht aus der Forschung stammt. In ersten Architektur-Blogs von AWS (Orban, 2017) werden LZ*s als Implementation einer der sogenannten *6R* verwendet (Orban, 2016). Bei diesen *R*s handelt es sich um die von Gartner, 2013 ursprünglich fünf erkannten Ansätze von Organisationen in die Cloud zu migrieren. Basierend auf diesen Ansätzen nennt Orban, 2017, dass als Teil der Migration das bestehende Applikations-Portfolio nach Bedürfnissen gruppiert werden soll, um passende Operations- und Application-Landing-Zones zu identifizieren. Die LZ* definieren hierbei high-level Architekturpatterns und Capabilities (Orban, 2017). Auf dieser ersten Beschreibung des Begriffs aufbauend werden die Dokumentationen aller Cloud-Plattformen konsultiert, um Ähnlichkeiten in der Definition feststellen zu können.

Microsoft Azure

Implizierte Definition LZ*: «Ein Azure Zielzone [...] bietet eine konsistente Möglichkeit, zum

[1] siehe Suche mit Query ('cloud landing zone' OR 'landing zone architecture' OR 'cloud foundation' OR 'multi-account architecture') AND ('design principles' OR 'best practices' OR 'architecture components' OR 'security' OR 'governance') auf https://swisscovery.slsp.ch/

Einrichten und Verwalten ihrer Azure-Umgebung [...]. Sie sorgt für Konsistenz in ihrer gesamten Organisation, indem es wichtige Anforderungen für Sicherheit, Compliance und Betriebseffizienz [...] erfüllt. Sie bieten eine [...] Grundlage, die auf [*Core*-]Designprinzipien in acht Designbereichen ausgerichtet ist. »(Microsoft, 2025)

Kommentar: Microsoft bietet eine Referenzarchitektur und unterteilt diese in Plattform- und Anwendungs-LZ*s. Es werden Designprinzipien definiert, welche die Grundlage für die Architektur bilden sollen. Microsoft, 2025

Amazon Web Services (AWS)

Implizierte Definition LZ*: «Eine landing zone ist eine gut strukturierte AWS Umgebung mit mehreren Konten, die skalierbar und sicher ist. Dies ist ein Ausgangspunkt, von dem aus Ihr Unternehmen schnell Workloads und Anwendungen starten und bereitstellen kann, ohne auf Ihre Sicherheits- und Infrastrukturumgebung verzichten zu müssen.»(AWS, 2026b)

Kommentar: AWS bietet die Möglichkeit LZ*s von AWS Control Tower oder eine eigene Implementation zu nutzen. Das LZ*-Feature von AWS Control Tower wird beschrieben als unternehmensweiter Container, der all Ihre Organisationseinheiten (OUs), Konten, Benutzer und andere Ressourcen enthält, für die Sie Compliance-Vorschriften gelten möchten. AWS, 2026b, AWS, 2026a

Google Cloud Platform (GCP)

Implizierte Definition LZ*: «[...] Eine Landing-Zone ist eine modulare und skalierbare Konfiguration, die es Organisationen ermöglicht, Google Cloud für ihre Geschäftsanforderungen zu übernehmen. Eine Landing-Zone ist oft eine Voraussetzung für die Bereitstellung von Unternehmens-Arbeitslasten in einer Cloud-Umgebung. Eine Landing-Zone ist keine Zone oder zonale Ressourcen. »(Google, 2026)

Kommentar: Google beschreibt die primären Elemente eine LZ* und bietet eine Beispielarchitektur. Es wird dem Kunden überlassen die LZ* zu gestalten. Google, 2026

IBM Cloud

Implizierte Definition LZ*: «Die *einsetzbaren Architekturen* sind ein vorkonfigurierter Satz von Infrastructure-as-Code-Assets (IaC IAC) »(IBM, 2026)

Kommentar: IBM sieht LZ*s als *einsetzbare Architekturen* und stellt konkrete Assets für Partnertechnologien bereit. IBM, 2026

Alibaba Cloud

Implizierte Definition LZ*: «Alibaba Cloud Landing Zone is a set of IT governance frameworks designed to guide enterprises to deploy on and migrate to Alibaba Cloud. »(Alibaba, 2026)

Kommentar: Alibaba sieht LZ*s als Frameworks, welche Unternehmen bei der Migration zu Alibaba Cloud unterstützen sollen. Alibaba, 2026

Die Analyse der verschiedenen Cloud-Plattformen zeigt, dass der Begriff LZ* nicht komplett einheitlich verwendet wird. Es gibt teilweise Überschneidungen mit anderen Begriffen wie *Foundation*, *Architecture* oder *Framework*. Trotz der Unterschiede wird sichtbar, dass die verschiedenen Cloud-Plattformen ähnliche Konzepte und Komponenten beschreiben, welche in der Literatur teilweise als Teil einer LZ* beschrieben werden. Um diese Beobachtung zu festigen, werden in Tabelle 3.3 ebenfalls die Komponenten der Cloud-Plattformen verglichen. Zum Vergleich werden folgende Komponenten verwendet: IAM / Account Management, Network, Resource Management, Compliance, Financial Management und Security.

Der Vergleicht zeigt, dass es durchaus Überschneidungen gibt bei den Komponenten welche die verschiedenen Anbieter als Teil einer LZ* beschreiben. Unterscheiden tun sich vor allem die Terminologie und die Art und Weise, in der die Komponenten beschrieben werden.

Aspekt / Anbieter	**AWS**	**Azure**	**GCP**	**AliBaba**	**IBM**
IAM / Account Management	ergibt sich aus Accounts und Roles and Permissions in AWS Control Tower	Identity and Access Management im Azure Cloud Adoption Framework. Umfasst Authentication, Access auf Ressourcen, Separation of Duties sowie Hybrid Identities mit Entra	vorhanden sowohl mit Google als Identity Provider wie auch mit 3rd Party Identity Providern	Role Authorization über Alibaba Cloud RAM, Integration mit Enterprise SSO zur einheitlichen Authentifizierung und Umsetzung des Least-Privilege-Prinzips	Läuft unter dem Aspekt Security
Network	Networking in AWS Control Tower mit Unterstützung von VPCs	Network topology and connectivity. Unterscheidung zwischen corp (interne Anwendungen) und online (Internet-exponierte Anwendungen)	nutzt ebenfalls VPCs und typischerweise eine Hub-and-Spoke Architektur	Planung von Enterprise-Netzwerken inkl. IP-Segmentierung, Network Control Policies und VPC-Zuweisung für verschiedene Accounts	vorhanden
Resource Management	über AWS Organizations mit Organisationseinheiten (OU) und Accounts	Resource organization mit Naming, Tagging, Subscription Design und Management Group Hierarchie	Strukturierung über Folders und Projects, gebunden an einen Root-Node	Ressource Management über Labels und Directory Permissions zur Integration mit Enterprise Account Systemen	läuft unter Service Management
Compliance	indirekt über Baselines und Guardrails in AWS Control Tower	Azure Governance stellt sicher, dass Designentscheidungen in Identity, Network, Security und Management eingehalten werden	vorhanden über das Compliance Resource Center	Cloud-Compliance Lösungen für Industry Norms und interne Policies (z.B. Encryption, Port Restrictions)	läuft ebenfalls unter Security

Aspect / Cloud	**AWS**	**Azure**	**GCP**	**AliBaba**	**IBM**
Financial Management	nicht als eigener Aspekt, sondern implizit über Billing der AWS Organization	Azure Billing und Microsoft Entra Tenant. Betrachtung verschiedener Angebotsmodelle wie Pay-As-You-Go	Cost efficiency and control über Google Cloud Billing	Kostenübersicht pro Branch / Department / Project über Account Labels zur Unterstützung von Cost Optimization	läuft unter Service Management
Security	eigener Security-Aspekt mit Compliance Validation, Data Protection, Resilience und Infrastructure Security	Security Design Area mit Fokus auf Security Operations Tooling und Zero Trust	eigener Security-Blueprint mit Best Practices	Enterprise Security Lösungen für Data, Storage, Host und Network Security	IAM, App-Security, Datensicherheit, Infra & Endpunkte, IDS, Governance
Quellen	AWS, 2026b, AWS, 2026a	Microsoft, 2025	Google, 2026	Alibaba, 2026	IBM, 2026

Tabelle 3.3: Vergleich zentraler Landing-Zone Aspekte zwischen Cloud Providern

3.3 Synthese eines abstrahierten Metamodells

Da die Definition von LZ* bei den verschiedenen Anbietern teilweise unterschiedlich ist, fehlt als Grundlage noch immer eine definitive Terminologie und Struktur, um die verschiedenen Komponenten einer LZ* zu beschreiben. Es wird dazu ein eigenes Metamodell erstellt, um eine klare Grundlage für die Arbeit zu schaffen. Dieses Metamodell soll zumindest im Rahmen dieser Arbeit und des Kontextes des Vorhabens eine LZ* sowie deren Beziehungen zueinander darstellen.

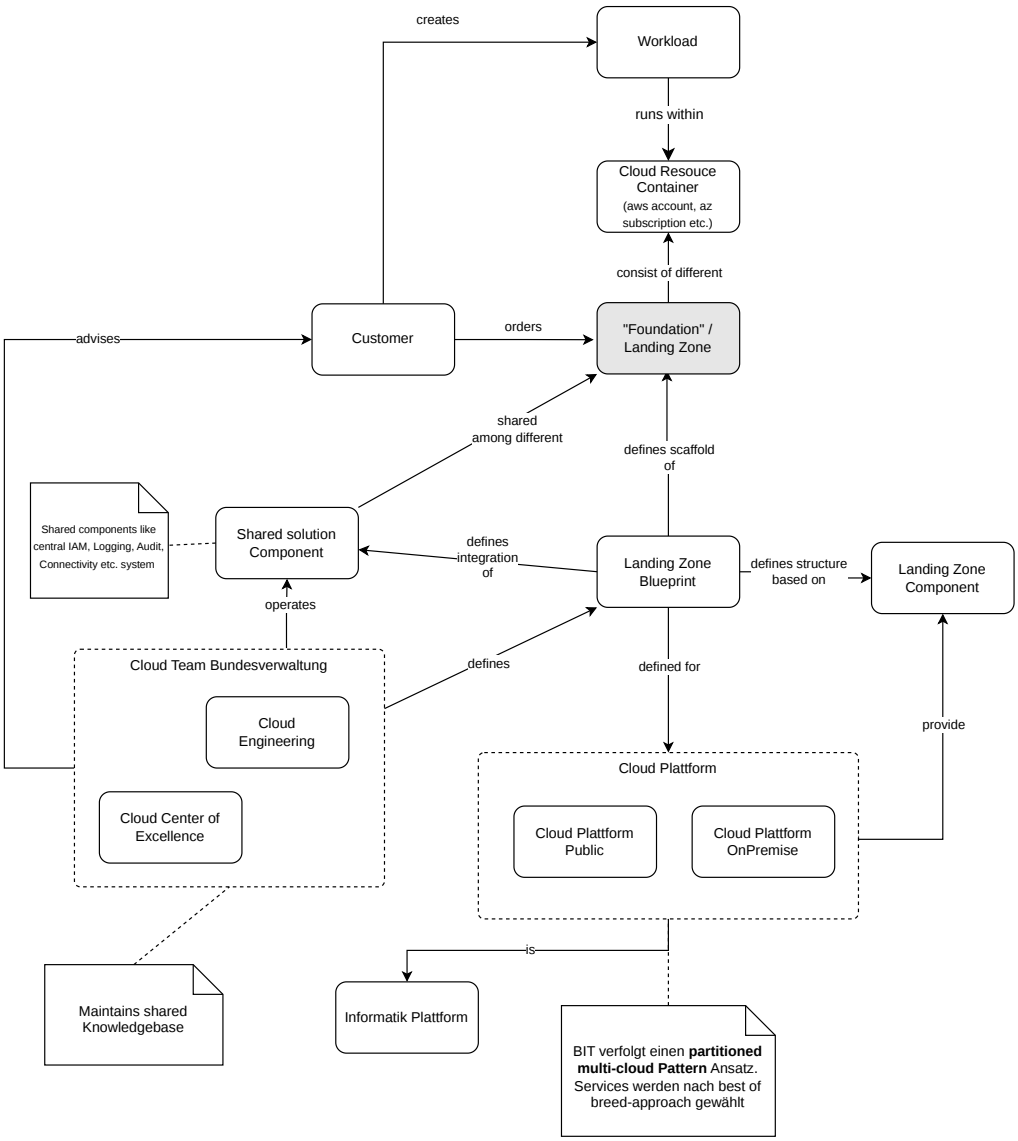


Abbildung 3.2: Metamodell für die LZ* Modellierung, eigene Abbildung

Abbildung 3.2 zeigt das erstellte Metamodell, welches die verschiedenen Elemente einer LZ* enthält, wie sie im Kontext dieser Thesis als *Foundation* verstanden wird. Diese wird auch zentral abgebildet, da sie direkt vom Kunden bestellt wird. Eine Foundation kann mehrere *Cloud Resource Containers* enthalten, welche die Cloud Ressourcen und somit den Workload des Kunden beinhalten. Der *LZ* Blueprint* ist ein Provider-spezifisches Template, welches die Cloud Resource Container mit den entsprechenden Konfigurationen und Governance Regeln definiert. Im LZ*

Blueprint sind ebenfalls die *Shared Solution Components* enthalten, welche geteilte Komponenten darstellen, die in der LZ* bereitgestellt werden. Diese können beispielsweise IAM, zentrale Logsammlung, Netzwerkkonnektivität etc. sein, die von mehreren Cloud-Resource-Containern geteilt werden.

Das Metamodell dient als Grundlage für die weitere Analyse und Entwicklung der Architektur-Varianten im Rahmen der Arbeit.

3.4 Erhebung von Landing-Zone-Fähigkeiten

Der Vergleich von den LZ*-Komponenten auf den verschiedenen Cloud-Plattformen zeigt, dass eine strukturelle Überlappung auf technischer Ebene existiert. Im Sinne des forschungsmethodischen Ansatzes wird angenommen, dass sich diese technische Ebene auf die Bedürfnisse der Organisation zurückführen lässt, welche sie mit der LZ* erfüllen möchte. TOGAF* hat für diese Verknüpfung die Methodik des Capability Mapping definiert. Capability Mapping dient dazu, die Fähigkeiten einer Organisation mit der technischen Implementation zu verknüpfen, um sicherzustellen, dass die Technologie die Capabilities der Organisation unterstützt (vpadmin, 2025). Für die Anwendung auf das BIT* werden deshalb Annahmen über die Capabilities getroffen, welche mit der LZ* erreicht werden sollen. Als Ausgangslage dient eine grobe Übersicht in Tabelle 3.4 mit Erwartungen an eine LZ*, welche vom BIT* basierend auf den Anforderungen der Stakeholder erstellt wurde.

Aspekt	**Beschreibung**
Bestellportal / Account Vending	Bestellung von Accounts / Landingzones
Konfiguration für Laufzeitumgebung	Bereitstellung und konfigurierbare Integration von Monitoring-, Logging-, Metrik- und Alerting-Funktionen auf Ebene der Landingzone zur laufenden Überwachung von Betrieb und Sicherheit inkl. automatischer Alarmierung und Kundentrennung.
Konfiguration für Verrechnungsstruktur	Konfigurationen für die Verrechnungselemente und Alerts zur Kostenkontrolle und Verrechnung. Zu jeder Landingzone wird ein Abrechnungselement konfiguriert, damit eine nachvollziehbare Leistungsabrechnung (Kostenisolation) aller verwendeten Ressourcen innerhalb einer Landingzone gemacht werden kann.
Sicherheitskonfigurationen	Sicherheitskonfigurationen, Einhaltung Grundschutz, Vulnerability Management, Zero Trust Architektur, Compliance Assessments, Netzwerksicherheit und regelmässige Sicherheitsaudits und Penetrationstests auf Ebene der Landingzone.
Identity & Access Management (IAM)	Rollen, Identitäten, Berechtigungen und Authentisierung-Verfahren. Zu jeder Landingzone gehört ein Administrator-Account. Dieser privilegierte Account definiert die Grenzen Landingzone, indem er Zugriff auf alle innerhalb der Landingzone vorgesehenen Konfigurations- und Verwaltungsmöglichkeiten bietet. Der Zugriff über diesen Account ist hoch privilegiert und ermöglicht die Verwaltung und Anpassung der gesamten Landingzone gemäss den definierten Governance- und Sicherheitsvorgaben.
Policy Management	Definition von Governance-Regeln, Tagging Policies, Ressourcenkonfigurationen und einheitliche Benennung und Klassifizierung aller Ressourcen für Auffindbarkeit, Compliance und Kostenmanagement für die Landingzones.

Dimension	**Beschreibung**
Nutzung von Services	Konfiguration, wie externe SaaS-Dienste oder Services sicher angebunden werden. Möglichkeit zur Einschränkung des Servicekatalogs auf Ebene der Landingzones.
Backup- und Recovery-Strategien	Vorkonfiguration von Backup-Lösungen und Notfallwiederherstellungskonzepten.
Netzwerk-Konnektivität	Einrichtung von Netzwerken (VPC, Subnets), Routing, VPNs oder Direct Connect/ExpressRoute, sowie Firewalls und Sicherheitsgruppen für sicheren Datenverkehr.
Verwendung des Anbieter-Marktplatzes	Der Marktplatz der Anbieter soll über alternative Beschaffungsgrundlagen genutzt werden können.

Tabelle 3.4: Stichwortartige Anforderungen der Stakeholder des BIT* an eine LZ*

Basierend auf den Anforderungen der Stakeholder werden mittels Prompt-Chaining zuerst Non-Functional Requirements (NFR) abgeleitet, mit denen die qualitativen Anforderungen für eine LZ* identifiziert werden sollen. Diese werden mit eigenen definierten NFR* für den Multi-Cloud-Kontext ergänzt. Anschliessend werden Level 1 Capabilities ermittelt, konsolidiert und darauf aufbauend Level 2 Capabilities abgeleitet. Die Levels stellen dabei eine Gruppierung und Detailgrad der Capabilities dar. Nach anschliessender Gegenprüfung erhalten wir in Abbildung 3.3 die Capability-Map. Zur Darstellung selbst wird hierbei die ArchiMate Capability-Map Notation verwendet. Zur Nachvollziehbarkeit der Capabilities in Abbildung 3.4, werden mit farbig gefüllten Post-Its die Level 1 Capabilities und mit farbig umrandeten Post-Its die Level 2 Capabilities detaillierter beschrieben. Jede der Capabilities kann potenziell als Startpunkt für eine Implementation auf tieferer Abstraktionsebene dienen. Somit kann sich sukzessive einer Business-zentrierten LZ*-Architektur angenähert werden.

Diese Arbeit beschränkt sich auf eine Auswahl von Capabilities, um den Prozess exemplarisch aufzuzeigen. Die Auswahl erfolgt hierbei durch Priorisierung mit dem BIT*. Die mit dem BIT* identifizierte grösste Herausforderung ist die Nachvollziehbarkeit der Anwendung von Regulationen auf die konkreten Cloud-Ressourcen. Eine strukturierte konzeptionelle Lösung für diese Problematik wird nachfolgend als Traceability-Architektur bezeichnet. Diese Architektur leitet sich aus den Level 1 Capabilities Security Governance, Policy and Compliance Lifecycle Governance, Audit & Evidence Governance sowie Configuration and Standards Governance ab. Basierend auf der Traceability-Architektur werden Use-Cases beschrieben. Diese Use-Cases dienen dazu den Scope der Implementierung der Traceability-Architektur eingrenzen zu können. Die Architektur selbst kann von verschiedenen Blickwinkeln aus betrachtet werden: zum einen die top-down Betrachtung des Compliance-Officers und zum anderen die bottom-up Betrachtung des Projektverantwortlichen. Ein Compliance Officer möchte nachvollziehen können, ob Regulationen und schlussendlich das Mapping auf konkrete Cloud-Ressourcen eingehalten wurde. Er weiss hierbei, welche Massnahmen basierend auf welchem Schutzbedarf ergriffen werden müssen. Ein Projektverantwortlicher kann den Schutzbedarf seiner Cloud-Ressourcen identifizieren, weiss jedoch nicht direkt, was dies für eine Cloud-Plattform impliziert. Ein Cloud-Experte ist weder über den Projektkontext im Klaren noch die Gesamtheit der regulatorischen Anforderungen, welche existieren. Der Cloud-Experte hat jedoch Domänenwissen über die Umsetzung von Sicherheitsmassnahmen auf einer gegebenen Cloud-Plattform.

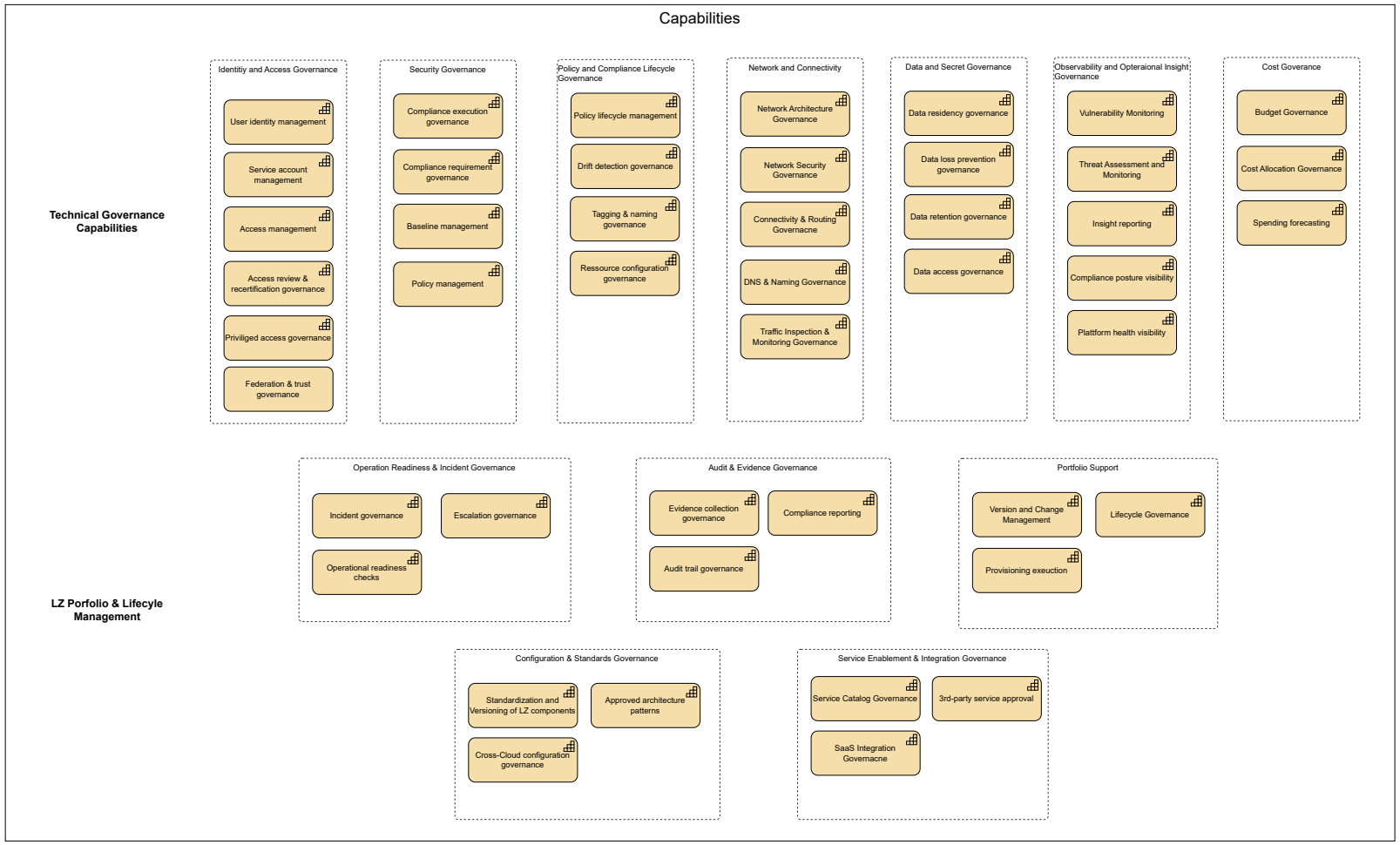


Abbildung 3.3: Capability Map mit Level 1 und Level 2, eigene Abbildung

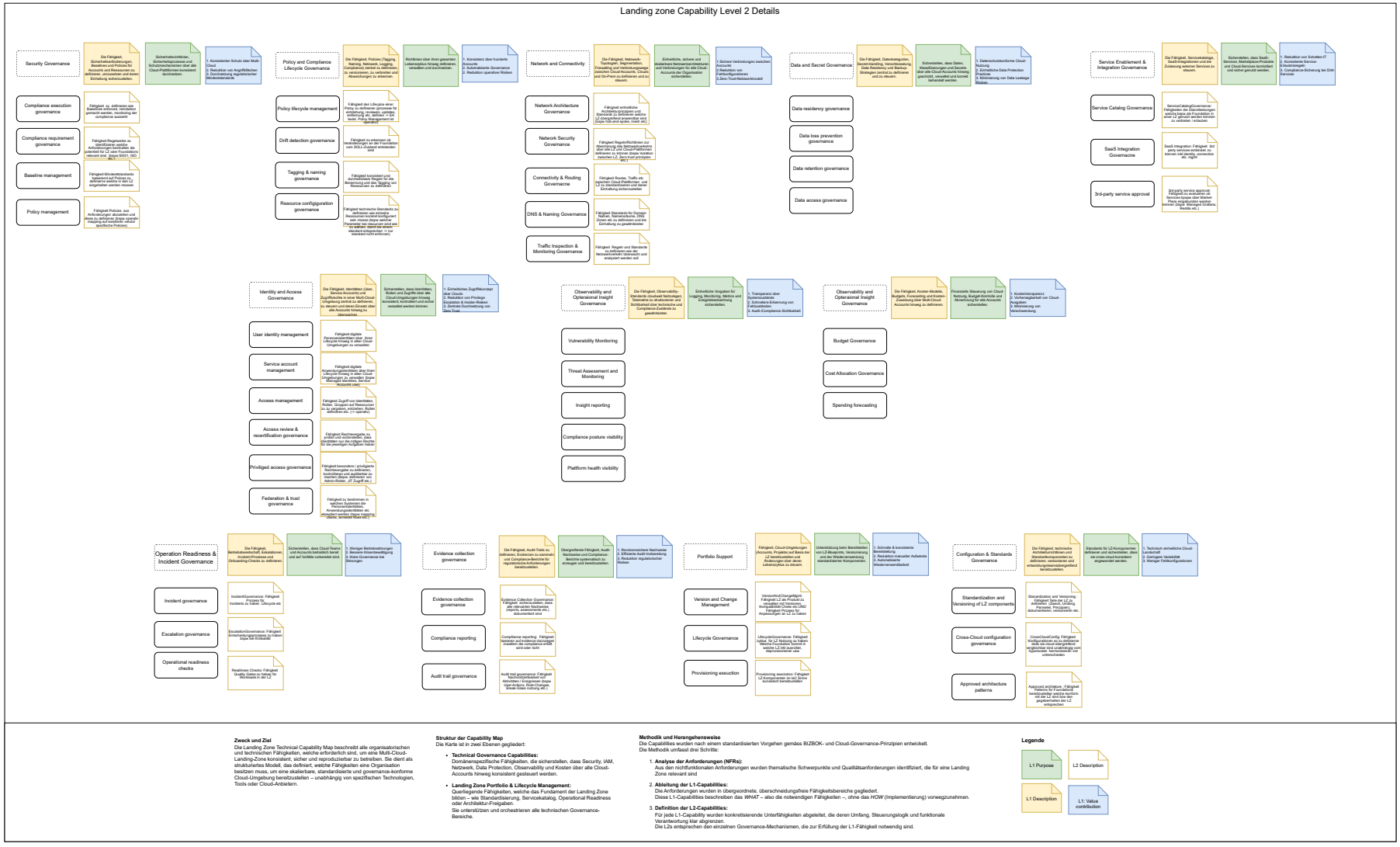


Abbildung 3.4: Capability Map mit Level 2, eigene Abbildung

Mit dem Fokus auf die Traceability-Architektur ergeben sich in Abbildung 3.5 die Use-Cases. Mit UC1 und UC2 definiert und konkretisiert der Compliance-Officer die relevanten regulatorischen Anforderungen für bspw. die Organisation des BIT*. Der Cloud-Expert implementiert anschliessend die technischen Massnahmen (z.B. identifiziert / gruppiert Policies, etc.). Der Projektverantwortliche kann schlussendlich seinen Schutzbedarf identifizieren und mit dieser Grundlage theoretisch dann auch seine LZ* bestellen. Die Verantwortlichkeiten der Akteure werden folgendermassen definiert.

- Der **Compliance Officer** verantwortet den Audit von Projekten innerhalb der Organisation. Der Akteur ist potenziell dem CIO angehängt.
- Der **Cloud-Experte** unterstützt Vorhaben beim Aufbaue ihres Projektes in einer der Cloud-Plattformen. Der Akteur ist potenziell Teil eines Cloud-Center-of-Excellence.
- Der **Projektverantwortliche** verantwortet ein Projekt und damit auch die Nutzung von Cloud-Plattformen. Der Akteur ist potenziell in der Projektorganisation angegliedert.

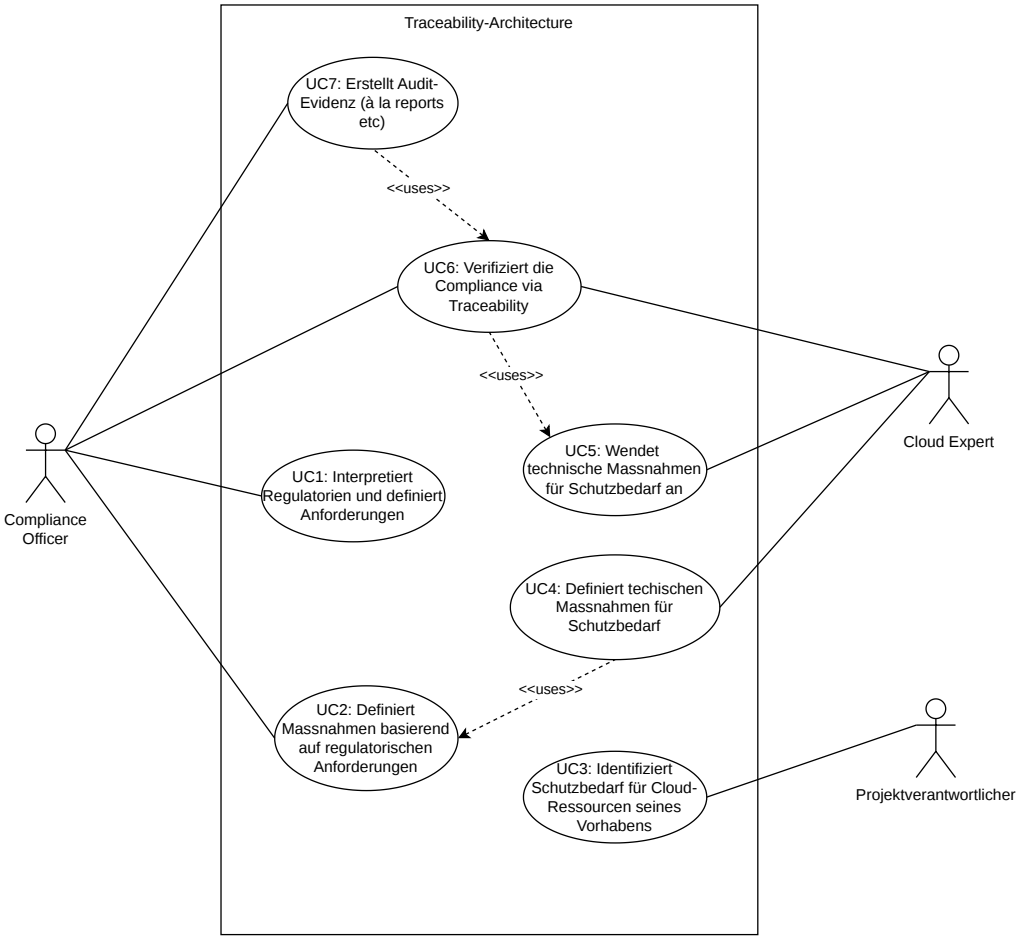


Abbildung 3.5: Use-Cases basierend auf Traceability-Architecture, eigene Abbildung

Die zentralen Use-Cases aus Abbildung 3.5 um eine technische Machbarkeit der Traceability Capability aufzuzeigen sind UC4 «Definiert technische Massnahmen für Schutzbedarf», um die Traceability-Struktur zu erzeugen und UC6 «Verifiziert die Compliance via Traceability»für die effektive Nutzung der Traceability-Struktur.

UC4 Definiert technische Massnahmen für Schutzbedarf	
Name	Definiert technische Massnahmen für Schutzbedarf
Beschreibung / Ziel	Der Use-Case beschreibt, wie abstrakte Governance Controls in konkrete Cloud-spezifische technische Massnahmen übersetzt werden. Ziel ist es, die Traceability-Beziehungen zwischen Controls und technischen Massnahmen zu modellieren, um eine prüfbare Grundlage für eine Compliance-Verifikation zu schaffen.
Akteure	Cloud-Experte, Compliance Officer
Vorbedingung	Regulatorische Anforderungen wurden interpretiert und in Schutzbedarfe und Controls überführt. Schutzbedarfskategorien sind definiert. Die Traceability-Architektur ist konzeptionell etabliert.
Nachbedingung	Für einen Schutzbedarf sind technische Massnahmen definiert. Mit den Massnahmen sind eindeutig Controls und Schutzbedarfskategorien verknüpft. Die Traceability-Beziehungen sind explizit modelliert.
Hauptablauf	1. Der Cloud-Experte analysiert den Schutzbedarf und zugehörige Controls 2. Der Cloud-Experte definiert geeignete technische Massnahmen basierend auf dem Schutzbedarf 3. Der Cloud-Experte ordnet die technischen Massnahmen den Ressourcentypen auf Cloud-Plattformen zu 4. Der Cloud-Experte modelliert die Traceability-Beziehungen. Dabei erfolgt die Zuordnung von Schutzbedarf zu Control, von Control zu technischen Massnahmen und von technischer Massnahme zu Ressourcentyp. 5. Der Compliance Officer prüft situativ, ob die definierten Massnahmen den regulatorischen Anforderungen entsprechen 6. Der Cloud-Experte gibt die definierten Traceability-Beziehungen frei 7. Das System speichert die technischen Massnahmen und die zugehörigen Traceability-Beziehungen als Artefakte
Alternative Abläufe	**Keine geeignete technische Massnahme identifizierbar** 1. Der Cloud-Experte identifiziert, dass keine geeignete technische Massnahme existiert 2. Der Cloud-Experte markiert den Schutzbedarf als nicht vollständig operationalisierbar 3. Das System kennzeichnet dies als Risiko / Lücke für die Compliance-Verifikation 4. Der Compliance Officer entscheidet über weitere Massnahmen bzw. Vorgehen
Ergebnis	Das System hält für einen Schutzbedarf mindestens eine technische Massnahme. Das System verfügt über Traceability-Beziehungen zwischen Schutzbedarf, Control und technischer Massnahme. Der Compliance Officer ist in der Lage, eine Compliance-Verifikation durchzuführen.

Tabelle 3.5: UC4 Definiert technische Massnahmen für Schutzbedarf

UC6 Verifiziert die Compliance via Traceability	
Name	Verifiziert die Compliance via Traceability
Beschreibung / Ziel	Der Use-Case beschreibt, wie der Compliance-Status von Projekten gegenüber regulatorischen Anforderungen mittels Traceability-Architektur überprüft wird. Ziel ist eine nachvollziehbare Bestimmung des Compliance-Status.
Akteure	Compliance Officer, Cloud-Experte, Projektverantwortlicher
Vorbedingung	Regulatorische Anforderungen wurden interpretiert und in Schutzbedarfe und Controls überführt. Das Projekt hat seinen Schutzbedarf identifiziert. Technische Massnahmen für den Schutzbedarf eines Projekts sind gemäss item 3.5 definiert und modelliert. Die Traceability-Architektur ist etabliert.
Nachbedingung	Der Compliance-Status ist in Form eines Artefaktes bestimmt. Die Traceability-Beziehungen sind einsehbar. Audit-fähiges Artefakt steht zur Verfügung.
Hauptablauf	1. Der Compliance Officer wählt ein Projekt zur Überprüfung aus. 2. Das System ermittelt die relevanten Traceability-Beziehungen zwischen Regulationen, Controls, technischen Massnahmen und Cloud-Ressourcen. 3. Das System wertet den Erfüllungsgrad der zugehörigen Controls aus. 4. Das System aggregiert die Ergebnisse zu einem Compliance-Status (compliant, non-compliant) aus Sicht der Regulation sowie aus Sicht des Projekts. 5. Das System stellt den Compliance-Status und die zugehörigen Artefakte dar. 6. Der Compliance Officer überprüft den Compliance-Status inkl. Trace / Artefakten. Der Cloud-Experte kann hierbei unterstützen.
Alternative Abläufe	**Unvollständige Traceability** 1. Das System identifiziert fehlende Traceability-Beziehungen. 2. Das System kennzeichnet den Compliance-Status. 3. Der Compliance Officer identifiziert die Massnahmen.
Ergebnis	Der Compliance-Status eines Projekts ist bestimmt, und ein Evidenz-Artefakt liegt vor. Die Traceability-Beziehungen sind einsehbar und auditfähig.

Tabelle 3.6: UC6 Verifiziert die Compliance via Traceability

3.5 Strukturierung der Evaluation der Architektur

Damit über die Einordnung der Traceability-Architektur selbst und der Varianten der Traceability-Architektur aus Unterabschnitt 4.1 (Traceability Architektur) eine Aussage über eine optimale Architektur (im Sinne der leitenden Fragestellung) gemacht werden kann, ist folgende Methodik vorgesehen:

1. Identifikation relevanter Bewertungsdimensionen für die Architektur selbst sowie für die Varianten
2. Recherche und Ableitung von Kriterien innerhalb der Bewertungsdimensionen (sowohl vom BIT* als auch aus der Literatur)
3. Ableitung eines Konformitätschecks der Traceability-Architektur an sich
4. Ableitung eines Bewertungsprozesses für die Varianten der Traceability-Architektur

Basierend auf der Orientierung an TOGAF* wird die Suche nach Dimensionen am ADM* Cycle ausgerichtet. Folgende Dimensionen werden ausgewählt:

Dimension	**Beschreibung**	**Begründung**
Architekturprinzipien	Prinzipien oder grundlegende Eigenschaften einer Architektur. Architekturprinzipien können eingehalten oder nicht eingehalten sein. Dies ist eine Evaluationsdimension, welche auf die Traceability-Architektur angewendet wird	Ergeben sich aus der Preliminary Phase im ADM* Cycle, in welcher die allg. Rahmenbedingungen gesetzt werden, von denen Architekturprinzipien ein Teil sind. Die Architekturprinzipien sind als Gatekeeper für Varianten gedacht
Qualitätsmerkmale	Qualitative Eigenschaften der Architektur analog zu klassischen NFR*. Qualitätsmerkmale können basierend auf einer Skala zu einem gewissen Grad erfüllt sein. Dies ist eine Evaluationsdimension, welche auf die verschiedenen Varianten der Architektur angewendet wird	Ergeben sich gedanklich aus den Phasen A-D im ADM* Cycle und dienen als vergleichende Kriterien zwischen verschiedenen Varianten
Entscheidung und Trade-Off	Spannungsfelder unterschiedlicher Architekturrichtungen, welche nicht per se richtig oder falsch sind. Trade-Off-Kriterien werden zum Labeling / zur Charakterisierung verwendet. Dies ist eine Evaluationsdimension, welche auf die verschiedenen Varianten der Architektur angewendet wird	Ergibt sich gedanklich aus den Phasen E-F im ADM* Cycle. Diese sollen die Eigenschaften der unterschiedlichen Varianten hervorheben, ohne wertend zu wirken, da die durch das Kriterium vergleichbaren Eigenschaften grundsätzlich keine negative oder positive Konnotation besitzen sollen
Kontext und Anwendung	Konkrete Rahmenbedingungen aus dem BIT* Use-Case, welche den Bewertungskontext der Evaluationskriterien bzw. Evaluationsdimensionen beeinflusst.	Ergibt sich aus der Festlegung des Kontextes in der Preliminary Phase und Ausarbeitung in den Phasen B-D im ADM* Cycle. Dient dazu im Rahmen des definierten Use-Cases zu beurteilen. Im Gegensatz zu den Architekturprinzipien des BIT* ist der Use-Case flexibler

Tabelle 3.7: Identifizierte Dimensionen des Evaluationsframeworks

Die Architekturprinzipien des BIT* ergeben sich aus den Vorgaben, die durch die Organisation selbst festgelegt sind. Von weiteren Kriterien des BIT* für diese Arbeit wird abgesehen. Die Architekturprinzipien aus Abbildung 3.6 gelten für das gesamte BIT*.

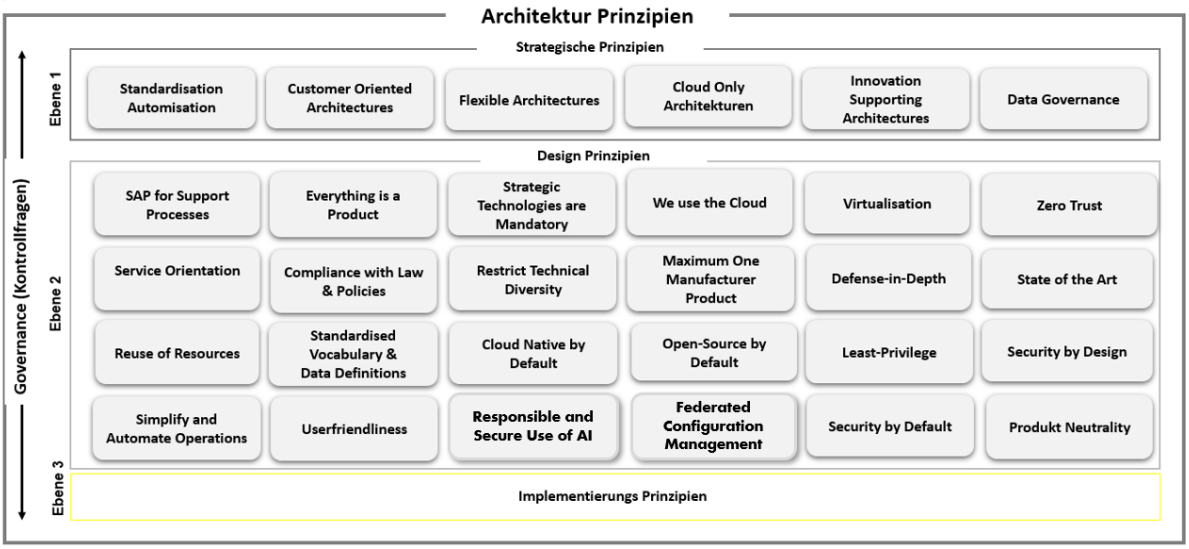


Abbildung 3.6: Architekturprinzipien des BIT* („Vorgabenportal BIT – Homepage“, 2025)

Für diese Arbeit werden die Prinzipien je nach gewähltem Use-Case selektiv berücksichtigt. Zu den Kriterien des BIT* sollen weitere identifiziert werden, die explizit für den Umstand einer multi-cloud-ready EA* vorgesehen sind. Hierfür werden initial die folgenden Recherchefragen formuliert:

«What are possible architecture principles for multi-Cloud ready enterprise architectures and solution architectures?», die mit Elicit AI recherchiert wurde (siehe Tools in Abschnitt 9 (Hilfsmittel)). Die identifizierten Prinzipien zeigen, dass die Frage mit dem Begriff «solution architecture» bereits zu stark auf konkrete technische Prinzipien fokussiert (siehe Appendix D (Report zu multi-cloud Architecture-Principles Literaturrecherche v1)). Dies entspricht nicht dem angedachten Abstraktionsniveau für Prinzipien einer EA*, weshalb die adaptierte Recherchefrage wie folgt lautet: «Which normative, technology-agnostic enterprise architecture principles must hold so that multiple cloud-specific solution architectures can be derived consistently?». Die identifizierten Architekturprinzipien in Tabelle 3.8 entstanden unter anderem basierend auf dem Screening von Abstracts nach Enterprise Architecture Principles und dem Bezug zu Cloud Computing Architecture bzw. Multi-Cloud Environments (siehe Appendix E (Report zu multi-cloud Architecture-Principles Literaturrecherche v2)).

Prinzip	Beschreibung	Quellen
Architectural Abtraction Principle	Architektur soll mittels Abstraktion businessspezifische Elemente von cloudspezifischen Implementationen loslösen. Durch konkrete technologische Elemente wie Containerisierung und Plattform-agnostische Interfaces kann somit eine Portabilität erreicht werden	Srikanth Gurram, 2025, Erl et al., 2013
Modular Decomposition Prinicple	Architektur soll Zerlegung von Cloud-Stacks in Komponenten gewährleisten. Dies soll eine Nutzung von Funktionen über Cloud-Tiers hinweg ermöglichen und eine starre Bindung an einen Anbieter verhindern	Papazoglou, 2012, Tritsiniotis, 2016
Reference Model Principle	Architektur soll auf Referenzmodellen aufbauen. Diese Modelle bilden eine gemeinsame Sprache für die architektonische Integrität und dienen als Leitfaden für die Identifizierung, Klassifizierung und Einführung von Technologien	Kratzke und Peinl, 2016, Zhang und Zhou, 2009
Governance Primacy Principle	Architektur soll klar auf Policies, Regulations und Unternehmensvorgaben aufbauen und nicht durch Technologie motiviert sein. Dies erreicht, dass trotz Unterschieden in den konkreten Lösungen / Cloud-Plattformen die Governancen als Input für das Vorgehen im Vordergrund bleibt	Zhang und Zhou, 2009, Gurram, 2025
Security Reconfigurability Principle	Security Architektur soll auf einer allgemeingültigen Security-Absicht aufbauen. Dieses dient wiederum als Ausgangslage für konkrete Security Models und deren spezifische Umsetzung pro Cloud-Plattformen. Absichten und Models, welche bereits zu stark auf eine Plattform fokussieren, stellen das Risiko dar, dass sie nicht auf einer anderen Cloud-Plattform anwendbar sind	Hajjat et al., 2010

Tabelle 3.8: Potenzielle Architekturprinzipien für eine multi-Cloud enabled EA* siehe Appendix E

Für die Qualitätsmerkmale werden sowohl eigens definierte Kriterien als auch jene aus der Recherche verwendet. Hierfür wird folgende Fragestellung verwendet: «Which architecture-related quality attributes are suitable for the comparative evaluation of enterprise architectures in technology-agnostic multi-Cloud environments?». Das Ergebnis gliedert sich in die folgenden Tiers: Tier 1 Universally Applicable, Tier 2 Context-Dependent Critical, Tier 3 Multi-cloud Specific, Tier 4 Emerging Considerations. Von Interesse sind hierbei vor allem die Kriterien, welche sich explizit auf den multi-Cloud Aspekt beziehen, sprich dem Tier 3.

Tier	Quality Attribute	Beschreibung im EA* Kontext	Quellen
1	Performance	Fähigkeit gewisse Antwortzeiten und Durchsatzraten zu gewährleisten sowie Fähigkeit auf Workload-Anpassungen zu reagieren	Papaioannou und Magoutis, 2013a, Bhargav Sai Pillati, 2025, Banijamali et al., 2019, Mohammad Asad Hussain, 2025
1	Reliability	Fähigkeit ein beständiges Serviceverhalten zu haben sowie Service Level Objectives einzuhalten. Fokus liegt auf der allgemeinen Robustheit, statt nur Ausfallsicherheit zu gewährleisten	Papaioannou und Magoutis, 2013a, Bhargav Sai Pillati, 2025, Banijamali et al., 2019
1	Cost-effectiveness	Fähigkeit Kosten transparent abzubilden um eine Vergleichbarkeit zu ermöglichen sowie Fähigkeit Möglichkeit zur Optimierung der Kosten zu geben - nicht nur zwangsläufig tiefstmögliche Kosten	Papaioannou und Magoutis, 2013a, Bhargav Sai Pillati, 2025, Menzel und Ranjan, 2011
2	Security	Fähigkeit Security-Ziele konsistent enforcen zu können sowie Fähigkeit Governancen und Policy-Enforcement zu ermöglichen	Bhargav Sai Pillati, 2025, Banijamali et al., 2019, Ghazaryan, 2025
2	Scalability	Fähigkeit mit Anpassungen in der Workload-, Daten, Nutzergrösse etc. umzugehen ohne strukturelle Änderungen vornehmen zu müssen	Bhargav Sai Pillati, 2025, Banijamali et al., 2019, Mohammad Asad Hussain, 2025
2	Availability	Fähigkeit eine gewisse Verfügbarkeit zu gewährleisten sowie Fähigkeit Redundanz-Strategien sowie High-Availability zu unterstützen	Banijamali et al., 2019, Ardagna et al., 2012
3	Interoperability	Fähigkeit Interaktion und Integration zwischen Systemen auf unterschiedlichen Plattformen zu erlauben	Banijamali et al., 2019, Joukov und Shorokhov, 2013
3	Portability	Fähigkeit der Architektur eine Migration oder Redeployment zwischen verschiedenen Plattformen zu erlauben, ohne grundsätzliches Redesign zu benötigen - bspw. ermöglicht durch Abstraktion	Mirsalari und Ranjbarfard, 2020
3	Vendor-Neutrality	Fähigkeit der Architektur strukturell von einem Anbieter und insbesondere von proprietären Funktionen abhängig zu sein	Bhargav Sai Pillati, 2025

Tier	Quality Attribute	Beschreibung im EA* Kontext	Quellen
4	Service Adaptability	Fähigkeit der Architektur Änderungen von Anforderungen, Interfaces oder dem Integrations-Kontext unterzubringen ohne die Architektur fundamental zu verändern. Service sind hierbei im Sinne der Capability zu verstehen, nicht ein konkreter technische Service Implementation	Bhargav Sai Pillati, 2025
4	Architectural Sustainability	Fähigkeit der Architektur sich langfristig an ändernde technologische sowie organisatorische Gegebenheiten anpassen zu können. Die Differenzierung zur Portability und Interoperability ist, dass diese auch längerfristig aufrechterhalten werden können	Bhargav Sai Pillati, 2025

Tabelle 3.9: Quality-Attriute aus Recherche nach Appendix F

Zu den externen Quality-Attributes kommen eigene Kriterien hinzu, die basierend auf der Traceability-Architecture definiert werden.

Quality Attribute	Beschreibung im Kontext der Traceability-Architecture	Begründung
Auditability	Fähigkeit, die beschreibt, wie nachvollziehbar die Zuweisung von Control und Assignment ist. Dies beinhaltet ebenfalls die verantwortlichen Akteure und Entscheidungspfade, die zum Assignment geführt haben.	Für die Generierung von Evidence für Governance und Compliance ist eine Auditierbarkeit essenziell
Automatability	Fähigkeit, dass Traceability-Beziehungen, Prüfungen und Nachweise automatisch erzeugt werden können. Dies beinhaltet ebenfalls die Zuweisung der Controls.	Automatisierung soll den manuellen Aufwand und die Fehleranfälligkeit der Architektur reduzieren. Bei hoher Komplexität und Anzahl Artefakte gewinnt dies umso mehr Relevanz
Maintainability	Aufwand, die Traceability-Modelle, Beziehungen und Controls aktuell zu halten	Policies auf den Plattformen und damit Zuweisungen zu entsprechenden Controls ändern sich regelmässig.
Adaptability	Fähigkeit, in der Architektur neue Traceability-Beziehungen, Artefakt-Typen, Controls etc. hinzuzufügen, ohne die Architektur grundlegend überarbeiten zu müssen	Cloud-Plattformen sowie relevante Regulatorien können sich ändern.

Quality Attribute	**Beschreibung im Kontext der Traceability-Architecture**	**Begründung**
Multi-Cloud Applicability	Grad der Fähigkeit, in der die Architektur über verschiedene Cloud-Plattformen konsistent anwendbar ist, ohne dass strukturelle Änderungen notwendig sind.	Es wird bereits vorausgesetzt, dass eine grundsätzliche Multi-Cloud-Fähigkeit besteht. Mit diesem Kriterium wird eine Vergleichbarkeit erreicht.
Complexity	Grad der zusätzlichen Vielschichtigkeit und Tiefe der Architektur. Es ist nicht die grundsätzliche Komplexität einer Architektur gemeint, sondern ein etwaiges Übermass an Komplexität in Anbetracht des Problems, das gelöst werden soll	Bei der Traceability handelt es sich bereits von Natur aus um eine vielschichtige Thematik. Entsprechend sollte vermieden werden, dass durch zu hohe Abstraktionen die Komplexität unverhältnismässig wird.
Traceability Strength	Grad, in dem die Beziehung zwischen Anwendung auf einer Cloud-Ressource und dem ausgehenden Control hergestellt werden kann	Es wird bereits vorausgesetzt, dass eine grundsätzliche Traceability besteht. Mit diesem Kriterium wird eine Vergleichbarkeit erreicht.

Tabelle 3.10: Dedizierte Quality-Attributes mit Fokus auf Traceability-Architecture

Für die Dimension der Trade-Off-Kriterien wird für die Recherche von Spannungsfeldern folgende Fragestellung verwendet: «Which architecture-level trade-offs shape the design and evaluation of enterprise architectures in technology-agnostic multi-Cloud environments».

Trade-Off Dimension	**Spannungsfeld**	**Quellen**
Performance vs. Portability	Optimieren der Runtime-Effizienz steht in Konflikt mit der langfristigen architektonischen Mobilität, da die Runtime-Effizienz häufig durch Optimierung auf ein bestimmtes Umfeld erreicht wird. Tendenz zu Performance führt zur bestmöglichen Ausnutzung von cloud-nativen Capabilities und tiefen Latenzen aber dafür verliert man an architektonischer Unabhängigkeit vom Provider und Vergleichbarkeit von Lösungen über Cloud-Plattformen hinweg.	Gibilisco, 2016, Emmanni, 2020, Papaioannou und Magoutis, 2013b
Cost effectiveness vs. Architectural specialization	Transparente Abbildung der Kosten sorgt für die Möglichkeit der Zuweisbarkeit von Kosten zu Architekturelementen. Dies erlaubt eine Vergleichbarkeit und das Senken des Risikos unerwarteter Kosten. Architecture specialization ermöglicht es wiederum spezialisierte / stark integrierte Services einzusetzen, welche die Kostenmodelle jedoch verkomplizieren. Eine Architektur die Kosten transparent hält, vermeidet tendenziell stark gekoppelte oder schwer kalkulierbare Strukturen	Arul, 2022, Sai und Pillati, 2025, Emmanni, 2020

Trade-Off Dimension	Spannungsfeld	Quellen
Flexibility vs. Structural simplicity	Flexibilität sorgt dafür, dass auf neue Requirements eingegangen werden kann und entsprechend eine bessere Anpassungsfähigkeit besteht. Gleichzeitig sinkt mit der steigenden Anzahl an architektonischen Konzepten und Optionen die Einfachheit der Architektur. Durch die Structural simplcity gibt es weniger Konzepte welche dafür klar definiert sind was zu einer besseren Verständlichkeit führt. Änderungen an Requirements manifestieren sich somit aber auch mehr in strukturellen Änderungen. Die genannten Quellen fokussieren vor allem auf Flexibility vs. Complexity. Da Complexity jedoch in Tabelle 3.10 jedoch als zusätzlicher Abstraktion beschrieben wurde, wurde hier Structural simplicity als Dimension gewählt	Gibilisco, 2016, Papaioannou und Magoutis, 2013a, Arul, 2022
Security vs. Usability	Security führt zu mehr Sicherheitsmechanismen, welche den Schutzgrad erhöhen sollen aber auch gleichzeitig zu operativen höheren Hürden führt. Mit der Usability wird die Architektur hingegen einfacher nutzbar und integrierbar. Es vereinfacht sich jedoch auch das Sicherheitsmodell bzw. Kontrollinstanzen	Berlato et al., 2020, Casola et al., 2018
Vendor Lock-In vs. Scalability	Scalability bzw. provider-native Optimierung führt zur Steigerung der Effizienz und Nutzung von Plattformspezifischer Skalierung, Verwaltung etc. Gleichzeitig erhöht sich jedoch die Abhängigkeit von bestimmten Anbieter. Vendor Lock-In Vermeidung hingegen bringt provider-neutrale Architekturen mit sich, welche die strategische Unabhängigkeit steigern. Mit dieser Steigerung der Austauschbarkeit sinkt jedoch die Nutzung von Provider-spezifischen Funktionen	Berlato et al., 2020, Quint und Kratzke, 2018, Zhao et al., 2022

Tabelle 3.11: Trade-Off Dimensionen aus der Recherche in Appendix G

Zusätzlich zu den Trade-Off-Dimensionen aus der Literatur werden die folgenden spezifischen Dimensionen definiert, die die Limitationen aus gewissen Architekturprinzipien bereits berücksichtigen und die sich explizit auf die Traceability-Architecture beziehen.

Trade-Off Dimension	Spannungsfeld	Begründung
Traceability Depth vs. Model Simplicity	Mit einer tiefen Traceability Depth werden mehr Artefakte modelliert und tiefere Traceability-Ketten generiert, womit Zusammenhänge besser rekonstruiert werden können. Die Model Simplicity führt dazu, dass das Metamodell überschaubarer bleibt, jedoch auch zum Verlust von Detailinformationen	Relevant basierend auf den Anforderungen, was für Evidenz / Artefakte erstellt werden soll

Trade-Off Dimension	**Spannungsfeld**	**Begründung**
Explicit Trace Modeling vs. Implicit Derivation	Mit explicit Tracing werden Beziehungen zwischen den Objekten im Metamodell bewusst modelliert, womit auch der initiale Modellierungsaufwand steigt. Dies führt zu einer besseren Prüfbarkeit und Auditierbarkeit. Die implicit Derivation leitet Traceability-Beziehungen aus ggf. bestehenden Regeln oder Strukturen ab, womit die Abhängigkeit von implizitem Wissen steigt.	Relevant wegen der Abhängigkeit, die sich aus der Modellierung von Cloud-Providern, aber auch aus dem Aufwand einer eigenen Modellierung ergibt
Central Traceability Model vs. Federated Traceability Models	Mit dem central Model wird ein einheitliches Model mit semantischer Vergleichbarkeit angestrebt, was eine einfachere domänenübergreifende Auswertung erlaubt. Mit dem federated Model existiert eine gewisse Vielfalt für eine Domäne, welche durch eine zentrale Auswertung gemapped werden muss. Die Konsistenz entsteht durch eine zentrale Auswertung / ein zentrales Mapping	Relevant je nach Heterogenität der Domänen, um keine künstlichen Modelle zu erhalten, die ggf. domänenspezifische Strukturen einschränken
Governance centric Traceability vs. Engineering centric Traceability	Eine Governance-centric Traceability setzt die Policies, Controls und Compliance in den Fokus. Die Traceability ist hierbei top-down angedacht. Der Fokus liegt hierbei hauptsächlich auf der Auditierbarkeit, was dem Fokus bspw. eines Compliance Officers entspricht. Die engineering-centric Traceability bringt die technische Nachvollziehbarkeit und die technischen Implementierungen / Artefakte in den Fokus. Dies entspricht potenziell der Betrachtung eines Entwicklers	Relevant für die Wahl der Betrachtungsrichtung, die gewählt wird.
Static Traceability Structure vs. Evolvable Traceability Structure	Eine static Structure ist bewusst in stabilen Strukturen angedacht, bei denen Änderungen nicht häufig passieren sollen. Dadurch wird auch langfristig eine Vergleichbarkeit erreicht. Die evolvable Structure bezieht sich auf die bewusste Anpassung und Erweiterung von Artefakt-Typen. Es wird somit bewusst offen definiert, um erweiterbar zu sein, was jedoch eine Vergleichbarkeit über die Zeit erschwert	Relevant für die Vergleichbarkeit von auf der Architektur generierten Artefakten wie Audits
High Formalization vs. Interpretative Flexibility	In einer Architektur mit high Formalization sind die Traceability-Konzepte oder Regeln klar definiert und maschinenlesbar. Hierdurch wird eine Automatisierbarkeit und Vergleichbarkeit erreicht, man verliert jedoch pragmatische Anpassungen an Domänen mit unscharfer Semantik oder Informationen. In einer Architektur mit interpretive Flexibility können Traceability-Beziehungen nicht vollständig formalisiert sein und semantische Lücken bewusst toleriert werden. Die Interpretation erfolgt manuell oder teilautomatisiert durch Menschen mit Kontextwissen	Relevant basierend auf dem Prozess, der angestrebt wird. Soll der Mensch aktiv in the loop sein oder wird eine volle Automatisierung angestrebt

Trade-Off Dimension	**Spannungsfeld**	**Begründung**
Traceability Coverage vs. Traceability Precision	Architekturen mit hoher Traceability Coverage zielen darauf ab, möglichst viele Artefakte, Beziehungen und Prozesse abzubilden. Dies führt zu hoher Transparenz und Vollständigkeit. Jedoch kann hierbei auch Overhead oder ein Rauschen entstehen, was die Interpretation erschwert. Bei Traceability Precision wird Wert darauf gelegt, die Beziehungen so präzise wie möglich darzustellen, um eine bessere Aussagekraft, klarere Prüfbarkeit und bessere Nachvollziehbarkeit zu erreichen.	Relevant für das Setzen des Fokus auf Breite oder Tiefe der Traceability je nach Anwendungsfall (z.B. Audit vs. Incident Investigation)
Policy-driven Traceability vs. Control-driven Traceability	Bei der Policy-driven Traceability wird von den Richtlinien, Standards und Governance-Vorgaben aus gearbeitet: *Was* muss nachgewiesen werden? Dies führt zu einheitlicher Auditierbarkeit, jedoch potenziell auch zu Überdokumentation und fehlender technischer Relevanz. Bei Control-driven wird von den verfügbaren Controls aus die Auditierbarkeit einer Anwendung aufgebaut. Dies bringt hohe technische Relevanz und im Engineering verankerte Auditierbarkeit, aber auch die Gefahr mit sich, nicht alle übergeordneten Regulatorien abzudecken. Eine Kombination der beiden Ansätze wäre möglich, muss jedoch im Voraus festgelegt werden.	Relevant für die Entscheidung, von welcher Sicht aus die Traceability aufgebaut werden soll

Tabelle 3.12: Selbst definierte Trade-Off Dimensionen

Die Dimension des Kontextes und der Anwendung, wie in Tabelle 3.7 beschrieben, stellt keine eigentliche Bewertungsdimension, sondern eine Gewichtungsdimension dar. Dies bedeutet, dass diese Dimension keinen Anspruch auf Objektivität erhebt und sich nicht zwangsläufig mit bestehender Literatur deckt. In dieser Arbeit werden als Kontext die definierten Use-Cases für das BIT* in Unterabschnitt 3.4 verwendet.

3.5.1 Konformitätsprozess für die Traceability-Architektur

Mittels der Dimension «Architekturprinzipien» wird ein Konformitätsprozess definiert. Der Prozess wendet die verschiedenen Prinzipien auf die Traceability-Architektur an, um eine Aussage über die Konformität machen zu können. Diese Aussage soll aufzeigen, dass die Struktur der Traceability-Architektur den Anforderungen aus der Dimension «Kontext und Anwendung» entspricht.

$$P = \{p_1, \ldots, p_n\} \tag{3.1}$$

$$P_{\text{selection}} = \{\, p \in P \mid p \text{ ist Teil der Auswahl } \} \tag{3.2}$$

(3.3)

(3.4)

Zur Fokussierung auf den Kontext, in welchem die Konformität gezeigt werden soll, wird eine Auswahl der Architekturprinzipien aus Abbildung 3.6 und Tabelle 3.8 vorgenommen. Diese Auswahl wird als $P_{\text{selection}}$ bezeichnet.

$$s_{\text{principles}} : P \rightarrow \{true, false\} \tag{3.5}$$

Die Erfüllung eines Architekturprinzips wird mit $s_{\text{principles}}(p)$ ermittelt.

$$s_{\text{principles}}(p) = \begin{cases} true & \text{falls Prinzip } p \text{ erfüllt ist} \\ false & \text{falls Prinzip } p \text{ nicht erfüllt ist} \end{cases} \tag{3.6}$$

$$passed = \forall p \in P_{\text{selection}} \ (s_{\text{principles}}(p) \text{ is true}) \tag{3.7}$$

Damit der Konformitätscheck als erfüllt gilt, müssen alle Architekturprinzipien in einem Mindestmass erfüllt sein ($passed = true$).

Nr	Beschreibung	Output
	Initialisierung	
1	Definieren vom Kontext und darauf aufbauend Leitgedanken, mit welchen die Bestimmung von Parametern hergeleitet werden kann. Diese können bspw. auf Use-Cases aufbauen	Beschreibung des Kontextes
2	Auswahl der Architekturprinzipien vornehmen aus Abbildung 3.6 und Tabelle 3.8	Liste der Auswahl an Architekturprinzipien $P_{\text{selection}}$
	Anwendung auf Traceability-Architektur	
3	Bewerten der Traceability-Architektur basierend auf allen Architekturprinzipien in $P_{\text{selection}}$ mit den Punkten aus $s_{\text{principles}}$	Bewertung $s_{\text{principles}}(p)$
4	Verfassen einer Gesamtbewertung basierend auf den vorherigen Ergebnissen	Gesamtbewertung der Variante

Tabelle 3.13: Konformitätscheck mit den einzelnen nummerierten Schritten

Für die Anwendung des Konformitätsprozesses wird in der Initialisierung die nachfolgende Auswahl $P_{\text{selection}}$ vorgenommen. Diese wird in folgende Gruppen eingeordnet, um bei der Beurteilung der Konformität verschiedene Themen kombinieren zu können:

- Gruppe 1: Governance-Driven Architecture
- Gruppe 2: Abstraction and Platform Independence
- Gruppe 3: Structural Architecture
- Gruppe 4: Security Architecture

Gruppe 1: Governance-Driven Architecture

A1: **Compliance with Law and Policies**

Wir beim BIT stellen sicher, dass wir die Gesetze, Vorschriften und Richtlinien einhalten, einschliesslich der Bundes-, Departements- und Amts-Prinzipien. **Begründung**: Die Nachvollziehbarkeit und das Enforcement von Regulationen ist einer der Beweggründe im Use-Case für die Traceability. **Quelle**: Auszug aus Abbildung 3.6

A2: **Data Governance**

Wir stellen sicher, dass wir als Leistungsbezüger BIT*-Daten mit klar definierten Regeln, Richtlinien und Praktiken verwalten, die die Qualität, Zugänglichkeit und Konsistenz der Daten gewährleisten. **Begründung**: Für das Placement der Foundation ist die Berücksichtigung der Datenklassifizierung ausschlaggebend. **Quelle**: Auszug aus Abbildung 3.6

Gruppe 2: Abstraction and Platform Independence

A3: **Cloud-native by Default**

Wir entwickeln und betreiben Cloud-native Anwendungen. **Begründung**: Der Lösungsansatz, welcher durch die Traceability-Architektur vorgeschlagen wird, soll sich bereits möglichst nah am Umfeld orientieren, in welchem er eingesetzt wird. **Quelle**: Auszug aus Abbildung 3.6, sowie Srikanth Gurram, 2025, Erl et al., 2013

A4: **Architectural Abstraction Principle**

Architektur soll mittels Abstraktion Business-spezifische Elemente von Cloud-spezifischen Implementationen loslösen. Durch konkrete technologische Elemente wie bspw. Containerisierung und Plattform-agnostische Interfaces kann somit eine Portabilität erreicht werden. **Begründung**: Um den unterschiedlichen Implementationen der Cloud-Provider gerecht zu werden ist eine Abstraktion für die Traceability unerlässlich. **Quelle**: Srikanth Gurram, 2025, Erl et al., 2013

A5: **Product Neutrality**

Die Sicherheitsvorgaben im BIT adressieren Anforderungen und keine Herstellerprodukte. **Begründung**: Herstellerspezifische Konzepte sollten bereits auf der Ebene der Architektur ausgeschlossen werden und höchstens konzeptionell betrachtet werden. **Quelle**: Auszug aus Abbildung 3.6

Gruppe 3: Structural Architecture

A6: **Serviceorientation**

Wir beim BIT stellen sicher, dass unsere Lösungen auf einem Design von Dienstleistungen basieren, welche die realen Geschäftsaktivitäten widerspiegeln und die Geschäftsprozesse des Unternehmens (oder zwischen Unternehmen) umfassen. **Begründung**: Die Traceability-Architektur soll sich an den verschiedenen Akteuren aus dem Use-Case ausrichten, um bereits eine optimale Ausrichtung auf potenzielle Anwender zu gewährleisten. **Quelle**: Auszug aus Abbildung 3.6

A7: **Modular Decomposition Principle**

Architektur soll Zerlegung von Cloud-Stacks in Komponenten gewährleisten. Dies soll eine Nutzung von Funktionen über Cloud-Tiers hinweg ermöglichen und eine starre Bindung an einen Anbieter verhindern. **Begründung**: Analog wie die Product Neutrality sollen nicht nur Konzepte, sondern auch die Komponenten der Architektur agnostisch gedacht werden. **Quelle**: Papazoglou, 2012, Tritsiniotis, 2016

A8: **Reference Model Principle**

Architektur soll auf Referenzmodellen aufbauen. Diese Modelle bilden eine gemeinsame Sprache für die architektonische Integrität und dienen als Leitfaden für die Identifizierung, Klassifizierung und Einführung von Technologien, siehe Tabelle 3.8. **Begründung**: Traceability selbst ist bereits ein abstrakter Begriff. Ein Metamodell, welches ein Mapping auf cloud-plattformspezifische

Namen und Konzepte ermöglicht, ist deshalb relevant. **Quelle**: Kratzke und Peinl, 2016, Zhang und Zhou, 2009

A9: **Standardisation Automation**

Wir streben danach, alle Bausteine unserer Architekturen so weit wie möglich zu standardisieren und die Prozesse so weit wie möglich zu automatisieren. Dieser Ansatz gilt für Geschäftsprozesse, Datenstrukturen, Daten, Anwendungen, Anwendungskomponenten und Infrastrukturkomponenten. **Begründung**: Die Automatisierung der Compliance-Prüfung ist einer der Hauptversprechen der Traceability-Architektur. Der Aufbau einer möglichst standardisierten Struktur für die Abbildung der Traceability und Evidences ist deshalb relevant. **Quelle**: Auszug aus Abbildung 3.6

Gruppe 4: Security Architecture

A10: **Security by Design**

Sicherheitsanforderungen an Software sowie Hardware werden bereits in der Entwicklung berücksichtigt, um spätere Schwachstellen in der Sicherheit zu verhindern. **Begründung**: Mit der Traceability selbst soll die Durchsetzbarkeit der Regulation wie Si001* erhöht werden. Dies gilt auch für die Konzeption der Traceability-Architektur als solcher. **Quelle**: Auszug aus Abbildung 3.6

A11: **Defense-in-Depth**

Es braucht mehrere komplementäre Massnahmen aus den Bereichen Personal, Technologie und Betrieb um eine Information effektiv zu schützen. **Begründung**: Da es sich bei der Traceability-Architektur selbst um ein Werkzeug für Audits der Foundations handelt, sind Sicherheitsüberlegungen noch wichtiger. **Quelle**: Auszug aus Abbildung 3.6

A12: **Security Reconfigurability Principle**

Security soll auf einer allgemeingültigen Security-Absicht aufbauen. Dies dient wiederum als Ausgangslage für konkrete Security Models und deren spezifische Umsetzung auf den Cloud-Plattformen. Absichten und Models, welche bereits zu stark auf eine Plattform fokussieren, stellen das Risiko dar, dass sie nicht auf einer anderen Cloud-Plattform anwendbar sind, siehe Tabelle 3.8. **Begründung**: Die Annahmen, auf welchen die Traceability-Architektur strukturiert sowie weitere Konzepte erarbeitet werden, sollten plattformübergreifend anwendbar sein. **Quelle**: Hajjat et al., 2010

3.5.2 Evaluationsprozess für die Varianten der Traceability-Architektur

Mittels der Dimensionen «Qualitätsmerkmale» und «Entscheidung und Trade-Off» wird ein Evaluationsprozess definiert. Der Evaluationsprozess wendet die verschiedenen Dimensionen an, um einen bewertenden Output generieren zu können. Durch Anwendung des Prozesses auf die verschiedenen Traceability-Architektur-Varianten soll eine Vergleichbarkeit ermöglicht werden. Schlussendlich kann somit eine Empfehlung im Anwendungskontext gegeben werden. Gedanklich orientiert sich dieser Prozess somit an einer Multiple-criteria decision analysis (MCDA), in der verschiedene Konfliktbereiche zueinander ins Verhältnis gesetzt werden.

Der Output des Prozesses ist der $Score_{\text{quality}}$, kombiniert mit einer textuellen Einordnung mittels der Trade-Off-Kriterien. Diese Trennung ergibt sich daraus, dass sich die Qualitätsattribute quantitativ, die Trade-Off-Kriterien jedoch nur qualitativ bewerten lassen. Mit dieser Trennung soll die Verständlichkeit der Bewertung erhöht werden. Des Weiteren soll betont werden, dass

die Trade-Off-Kriterien zur Einordnung des $Score_{\text{quality}}$ dienen. Als Score wird generell eine bewertende Grösse verstanden.

$$Q = \{q_1, \ldots, q_n\} \tag{3.8}$$
$$Q_{\text{selection}} = \{\, q \in Q \mid q \text{ ist Teil der Auswahl } \} \tag{3.9}$$
$$\tag{3.10}$$
$$T = \{t_1, \ldots, t_n\} \tag{3.11}$$
$$T_{\text{selection}} = \{\, t \in T \mid t \text{ ist Teil der Auswahl } \} \tag{3.12}$$
$$\tag{3.13}$$
$$Score_{\text{quality}} = \frac{\sum_{q \in Q_{\text{selection}}} w_{\text{quality}}(q)\, s_{\text{quality}}(q)}{\sum_{q \in Q_{\text{selection}}} w_{\text{quality}}(q)} \tag{3.14}$$
$$\tag{3.15}$$

Der $Score_{\text{quality}}$ ergibt sich aus einer Auswahl an Qualitätsattributen Q (siehe Tabelle 3.9 und Tabelle 3.10), welche als $Q_{\text{selection}}$ bezeichnet wird. Die Anzahl an Qualitätsattributen ist durch $|Q_{\text{selection}}|$ gegeben. Zur Gewichtung der einzelnen Merkmale wird $w_{\text{quality}}(q)$ verwendet.

$$w_{\text{quality}} : Q \to \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\} \tag{3.16}$$
$$s_{\text{quality}} : Q \to \{1, 2, 3, 4\} \tag{3.17}$$

Die Bewertung des $Score_{\text{quality}}$ erfolgt basierend auf einer Auswahl der Trade-Off-Kriterien T (siehe Tabelle 3.11 und Tabelle 3.12), welche als $T_{\text{selection}}$ bezeichnet wird.

Mit den beschriebenen Elementen wird folgender Evaluationsprozess definiert:

Nr	Beschreibung	Output
	Initialisierung	
1	Definieren vom Kontext und darauf aufbauend Leitgedanken, mit welchen die Bestimmung von Parametern hergeleitet werden kann. Diese können bspw. auf Use-Cases aufbauen	Beschreibung des Kontextes
2	Auswahl der Qualitätsattribute aus Tabelle 3.9 und Tabelle 3.10 vornehmen. Anschliessend jedem Qualitätsattribut ein Gewicht v_i entsprechend dem Definitionsbereich zuweisen	Liste der ausgewählten Qualitätsattribute $Q_{\text{selection}}$ mit Gewichten $w_{\text{quality}}(q)$
3	Auswahl der Trade-Off Kriterien vornehmen aus Tabelle 3.11 und Tabelle 3.12. Diese werden nicht gewichtet.	Liste der Auswahl an Trade-Off Kriterien mit $T_{\text{selection}}$
	Anwendung auf Traceability-Architektur-Variante	
4	Bewerten der Variante basierend auf allen Qualitätsattributen in $Q_{\text{selection}}$ mit den Punkten aus s_{quality}	Bewertung $s_{\text{quality}}(q)$

Nr	Beschreibung	Output
5	Berechnen des $Score_{\text{quality}}$	Berechneter Score $Score_{\text{quality}}$
6	Bewerten der Variante basierend auf den Trade-Off-Kriterien $T_{\text{selection}}$ mittels Punkten aus s_{tradeoff}	Bewertung $s_{\text{tradeoff}}(t)$ mit der textuellen Begründung
7	Verfassen einer Gesamtbewertung basierend auf den vorherigen Ergebnissen	Gesamtbewertung der Variante

Tabelle 3.14: Evaluationsprozess mit den einzelnen nummerierten Schritten

Für die Anwendung des Evaluationsprozesses wird in der Initialisierung folgende Auswahl für die Qualitätsattribute $Q_{\text{selection}}$ mit den Gewichten $w_{\text{quality}}(q)$ getroffen.

ID	$w(q)$	Qualitätsattribut	Begründung	Quelle
Q1	0.3	**Performance** Fähigkeit, gewisse Antwortzeiten und Durchsatzraten zu gewährleisten sowie auf Workload-Anpassungen reagieren zu können	Die Performance ist weniger relevant, da die APIs und Services der Cloud-Provider selbst eine gewisse Durchlaufzeit für die verschiedenen Aktionen haben. Ein Beispiel ist hierbei die Zeitdauer von der Anwendung einer Policy bis zum Erhalt des ersten Resultats	Papaioannou und Magoutis, 2013a, Bhargav Sai Pillati, 2025, Banijamali et al., 2019, Mohammad Asad Hussain, 2025
Q2	1	**Portability** Fähigkeit der Architektur eine Migration oder Redeployment zwischen verschiedenen Plattformen zu erlauben, ohne grundsätzliches Redesign zu benötigen - bspw. ermöglicht durch Abstraktion	Sehr relevant, da die Cloud-Plattformen im Use-Case (UC) des BIT noch nicht feststehen. Die betrachteten Plattformen Azure und AWS werden lediglich als Beispiele verwendet. Eine Anwendbarkeit auf weitere Cloud-Plattformen ist jedoch essenziell	Mirsalari und Ranjbarfard, 2020
Q3	0.6	**Auditability** Fähigkeit, die beschreibt, wie nachvollziehbar die Zuweisung von Control und Assignment ist. Dies beinhaltet ebenfalls die verantwortlichen Akteure und Entscheidungspfade, welche zum Assignment geführt haben	Die Nachvollziehbarkeit ist relevant, jedoch hauptsächlich für die Traceability selbst und weniger, wann die Beziehungen geändert / angepasst wurden. Die Gewichtung ist deshalb tiefer	Tabelle 3.10

ID	$w(q)$	Qualitätsattribut	Begründung	Quelle
Q4	1	**Automatability** Fähigkeit, dass Traceability-Beziehungen, Prüfungen und Nachweise automatisch erzeugt werden können. Dies beinhaltet ebenfalls die Zuweisung der Controls	Die Fähigkeit der Automatisierung ist einer der Hauptgründe für die Relevanz der Traceability	Tabelle 3.10
Q5	0.8	**Maintainability** Aufwand, die Traceability-Modelle, Beziehungen und Controls aktuell zu halten	Um eine langfristige Nutzung der Traceability-Architektur zu erlauben, ist ein geringer Maintenance-Aufwand wichtig. Dadurch wird bspw. die Aktualität der Traceability-Beziehungen ermöglicht	Tabelle 3.10
Q6	0.6	**Adaptability** Fähigkeit, in der Architektur neue Traceability-Beziehungen, Artefakt-Typen, Controls etc. hinzuzufügen, ohne die Architektur grundlegend überarbeiten zu müssen	Die Architektur selbst soll zwar um weitere Traceability-Artefakte erweiterbar sein, kann sich aber möglicherweise nicht beliebig allen Eigenheiten von Cloud-Plattformen beugen	Tabelle 3.10 und Bhargav Sai Pillati, 2025
Q7	0.8	**Complexity** Grad der zusätzlichen Vielschichtigkeit und Tiefe der Architektur. Es ist nicht die grundsätzliche Komplexität einer Architektur gemeint, sondern ein etwaiges Übermass an Komplexität in Anbetracht des Problems, welches gelöst werden soll	Der Inhalt der Modellierung ist möglicherweise komplex, jedoch sollte die Abstraktion gering bleiben, um eine Anwendung zu erleichtern. Entsprechend die höhere Gewichtung	Tabelle 3.10
Q8	1	**Traceability Strength** Grad, in dem die Beziehung zwischen einer Anwendung auf einer Cloud-Ressource und dem ausgehenden Control hergestellt werden kann	Der Informationsgehalt, welcher aus den Traceability-Beziehungen gezogen werden kann, ist der treibende Faktor für eine Traceability-Architektur. Deshalb ist dieses Attribut hoch gewichtet	Tabelle 3.10

Tabelle 3.15: Auswahl an Quality-Attributes

Ebenfalls wird eine Auswahl an Trade-Off-Kriterien $T_{\text{selection}}$ vorgenommen.

ID	Trade-Off Dimension	Begründung	Quelle
T1	Explicit Trace Modeling vs. Implicit Derivation	Eine der relevantesten Dimensionen, da sie hinterfragt, ob eine Nutzung von eigenen Strukturen zusätzlich zu jenen der Cloud-Provider sinnvoll ist. Dies ist im Kontext des BIT* relevant, um eine möglichst genaue Aussage für ein Audit machen zu können	Tabelle 3.12
T2	Governance centric Traceability vs. Engineering centric Traceability	Relevant aus Sicht der Akteure, um zu hinterfragen, aus welcher Perspektive die Modellierung am sinnvollsten ist. Dies ist im Kontext des BIT* relevant, da zu ermitteln ist, für welche Abteilung, der der Akteur angehört, eine Modellierung am wertvollsten ist	Tabelle 3.12
T3	Traceability Depth vs. Model Simplicity	Relevant für den BIT*-Kontext, um das Mass an Informationen zu ermitteln, das nötig ist, um ein Audit durchführen zu können	Tabelle 3.12
T4	Central Traceability Model vs. Federated Traceability Models	Für das BIT* mit verschiedenen Stakeholdern ist eine gewisse Öffnung/Flexibilität eines Modells potenziell relevant, um eine Nutzung zu fördern	Tabelle 3.12

Tabelle 3.16: Auswahl an Trade-Off-Kriterien

3.6 Proof-of-Concept zur Veranschaulichung

Die technische Machbarkeit der angedachten Architektur-Varianten aus Unterabschnitt 4.1 (Traceability Architektur) soll exemplarisch mit einem PoC* gezeigt werden. Es wird folgendes Vorgehen angewendet.

1. Definieren von didaktischen Kriterien
2. Bewerten der Architektur-Varianten mit den Kriterien
3. Beschreibung des Umfangs des PoC* basierend auf der Architektur-Variante

Die Wahl der Variante orientiert sich hierbei an didaktischen Kriterien und soll nicht die Evaluation der verschiedenen Architekturen aus Abschnitt 6 (Evaluation der Traceability Architektur) vorwegnehmen. Der Fokus liegt somit darauf, den grundlegenden Mechanismus der Traceability demonstrieren zu können und die Machbarkeit zentraler Aspekte aufzuzeigen.

Kriterium	**Beschreibung**	**Didaktische Relevanz**
Darstellung von Traceability Relationships	Die Architektur-Variante erlaubt eine verständliche Darstellung der Traceability-Beziehungen zwischen den regulatorischen Anforderungen und den architektonischen Artefakten des Metamodells. Die Bewertung erfolgt von 1 als unverständliche Darstellung bis 4 als verständliche Darstellung. 4 gilt als erstrebenswerte Ausprägung	Zur Demonstration sollten die Beziehungen klar sichtbar sein. Implizite Beziehungen oder Zusammenhänge machen die Versinnbildlichung abstrakter
Explizite Abbildung von Verantwortlichkeiten	Die Architektur-Variante veranschaulicht die Zuordnung von Verantwortlichkeiten in der Architektur und den beteiligten Rollen wie bspw. Akteuren aus dem Use-Case-Diagramm in Abbildung 3.5 (Use-Cases basierend auf Traceability-Architecture, eigene Abbildung). Die Bewertung erfolgt von 1 als implizite Abbildung bis 4 als explizite Abbildung. 4 gilt als erstrebenswerte Ausprägung	Der Use-Case der Traceability basiert auf der Interaktion zwischen den verschiedenen Rollen, sprich auf dem Identifizieren der Regulation, dem Definieren der technischen Umsetzung und der Anwendung. Deshalb ist es relevant, die Sichtbarkeit dieser Interaktion zu demonstrieren
Reduzierte strukturelle Komplexität	Die Architektur-Variante soll eine begrenzte strukturelle Komplexität besitzen und sich auf wesentliche architektonische Konzepte der Traceability-Architektur/des Metamodells beschränken. Die Bewertung erfolgt von 1 als hohe Komplexität und 4 als geringe Komplexität. 4 gilt als erstrebenswerte Ausprägung	Der PoC* soll zur Beleuchtung der zentralen Konzepte dienen und kein Minimum Viable Product (MVP) sein. Durch eine reduzierte Komplexität ist die Traceability-Logik einfacher nachzuvollziehen

Kriterium	**Beschreibung**	**Didaktische Relevanz**
Alignment mit Traceability Use-Case	Die Architektur-Variante deckt den Traceability Use-Case ab, ohne für den Use-Case irrelevante Konzepte/Implementationen einzuführen. Die Bewertung erfolgt von 1 als aufwändige Abbildung bis 4 als pragmatische Abbildung. 4 gilt als erstrebenswerte Ausprägung	Der PoC* soll durch die Fokussierung auf den Use-Case seinen Fokus behalten und somit nicht ausufern

Tabelle 3.17: Didaktische Kriterien für die Auswahl der Variante der Traceability-Architecture für den PoC*

Basierend auf den Kriterien und der Bewertungsskala wird die Bewertung der verschiedenen Architektur-Varianten gewählt. Die Skalen sind jeweils so definiert, dass 4 als erstrebenswerte Ausprägung gilt, was jedoch nicht bedeutet, dass diese erreichbar ist.

Bewertung *Darstellung von Traceability-Beziehungen*		
Kriterium	**Bewertung**	**Begründung**
Control-centric	4	Ideale Darstellung aufgrund des Grundgedankens der Architektur-Variante, die Anforderungen aus den Regulationen nach Controls zu gruppieren
Component-centric	3	Beziehungen sind darstellbar, leiten sich jedoch indirekt über die Struktur nach LZ* Components ab. Sprich, man muss erst die Components verstehen, und dann ergibt sich die Traceability
Category-centric	2	Beziehungen sind indirekt über die Kategorien und wiederum die LZ* Components absehbar. Die schlechtere Sichtbarkeit ergibt sich direkt aus dem Design-Gedanken, ein Bundling via LZ*-Kategorien zu machen. Dadurch, dass der Fokus auf dem Labeling nach Kategorien liegt, ist es jedoch aus Sicht der Projektverantwortlichen einfach nachzuvollziehen.
Provider-native	1	Die Nutzung von Provider-nativen Tools ermöglicht zwar theoretisch eine Darstellung der Traceability, jedoch nur, wenn man die technischen Implementationen der Plattformen einbezieht. Dadurch ergibt sich eine hohe Implizität der Abbildung der Traceability-Beziehungen

Tabelle 3.18: Bewertung des Kriteriums *Darstellung von Traceability Relationships* aus Tabelle 3.17 (Proof-of-Concept zur Veranschaulichung)

Bewertung *Explizite Abbildung von Verantwortlichkeiten*		
Kriterium	**Bewertung**	**Begründung**
Control-centric	4	Verantwortlichkeiten sind direkt an Akteure gebunden: Compliance Officer mit Definition von Controls und Cloud-Experte mit technischer Umsetzung basierend auf den Controls

Bewertung *Explizite Abbildung von Verantwortlichkeiten*		
Kriterium	**Bewertung**	**Begründung**
Component-centric	3	Verantwortlichkeiten werden über die technischen LZ* Components geregelt. Die Verantwortung für die Components ist jedoch nicht eindeutig
Category-centric	2	Verantwortung ist abstrahiert über die LZ* Kategorien. Die Akteure verschwinden somit hinter der Kategorie / Zuordnung ist nicht direkt ersichtlich
Provider-native	1	Durch Nutzung von Plattformmechanismen ist die Rolle des Compliance Officers in der Architektur quasi nicht mehr sichtbar. Die Wahrnehmung ist, dass eine Verschiebung hin zum Cloud-Experten stattfindet.

Tabelle 3.19: Bewertung des Kriteriums *Explizite Abbildung von Verantwortlichkeiten* aus Tabelle 3.17 (Proof-of-Concept zur Veranschaulichung)

Bewertung *Reduzierte strukturelle Komplexität*		
Kriterium	**Bewertung**	**Begründung**
Control-centric	3	Die Architektur hat ein klares Ordnungsprinzip mit den Controls. Gleichzeitig ist dadurch jedoch die Anzahl an Traceability-Beziehungen höher
Component-centric	3	Struktur ist technisch intuitiv wegen dem Fokus auf die LZ* Components, womit aber die Traceability-Beziehungen auch über die Components laufen. Für ein Verständnis der Traceability muss somit die technische Beziehung verstanden werden.
Category-centric	4	Die LZ* Kategorien bündeln viele Details, wodurch die Architektur übersichtlich wirkt. Diese Übersichtlichkeit wird jedoch durch Abstraktion im Hintergrund gewonnen
Provider-native	2	Die strukturelle Komplexität der Provider-spezifischen Services erschwert es, ein einheitliches Ordnungsprinzip zu etablieren. Dadurch ist die Nachvollziehbarkeit geringer

Tabelle 3.20: Bewertung des Kriteriums *Reduzierte strukturelle Komplexität* aus Tabelle 3.17 (Proof-of-Concept zur Veranschaulichung)

Bewertung *Alignment mit Traceability Use-Case*		
Kriterium	**Bewertung**	**Begründung**
Control-centric	3	Der Fokus auf die Controls bildet bereits einen wichtigen Teil der Use-Cases ab, da der Fokus vom Compliance Officer ebenfalls hauptsächlich auf der Anwendung der Controls liegt.
Component-centric	3	Der Fokus auf die LZ* Components erfordert zuerst eine Ableitung über die technischen Components und erst anschliessend treten die Controls hervor, welche im Use-Case Ablauf beschrieben werden.

Bewertung *Alignment mit Traceability Use-Case*		
Kriterium	**Bewertung**	**Begründung**
Category-centric	2	Die Usability, welche durch die LZ*-Kategorisierung erreicht wird, sorgt bei der Abbildung für den Use-Case für eine starke Vereinfachung. Dadurch tritt die Traceability-Logik in den Hintergrund
Provider-native	2	Die Provider-spezifischen Implementierungen lenken den Fokus von der Abbildung der Traceability-Beziehungen und somit von den definierten Use-Cases weg.

Tabelle 3.21: Bewertung des Kriteriums *Alignment mit Traceability Use-Case* aus Tabelle 3.17 (Proof-of-Concept zur Veranschaulichung)

Basierend auf der Evaluation der Kriterien wird im PoC* die Variante der Component-centric Architektur (Variante 2) verwendet. Eine weitere Überlegung zu den bereits evaluierten Kriterien ist, dass die Aufteilung in die LZ*-Komponenten gewissermassen auch einer Know-how-Strukturierung bspw. im BIT* entsprechen könnte. Gemeint ist die Aufteilung bspw. in Networking, Storage, IAM* etc., welche (so die Annahme) teilweise von verschiedenen Abteilungen im BIT* verantwortet werden. Der PoC* soll somit neben der Funktionsweise der Traceability-Architecture zugleich den verschiedenen Ausprägungen des Cloud-Experten (durch die verschiedenen Abteilungen) gerecht werden.

Der PoC* soll den Fokus auf die Traceability-Architecture gewährleisten. Der Scope erhebt somit nicht den Anspruch einer ganzheitlichen Implementierung, sondern den, dedizierte Aspekte hervorzuheben. Um gleichwohl im Sinne der didaktischen Ziele eine gute Veranschaulichung zu erhalten, wird ein GUI-Prototyp für die Komponente namens Governance-Cockpit generiert. Dieser Prototyp (siehe Appendix A (Mock-Up zur Veranschaulichung des PoCs)) soll aufzeigen, wie sich ein Tool für die Akteure aus Tabelle 3.6 (Erhebung von Landing-Zone-Fähigkeiten) verhalten würde, welches auf einem System mit der Traceability-Architecture aufbaut. Das Ziel ist somit die Nutzerzentrierung, sprich den Fokus auf die Use-Cases der Organisation sicherzustellen. Die wesentlichen Schritte aus dem Hauptablauf des Use-Cases sind hierbei: die Auswahl des Projektes durch den Compliance Officer, wie in Abbildung 3.7, und die Überprüfung des Compliance-Status inkl. Trace, wie in Abbildung 3.8.

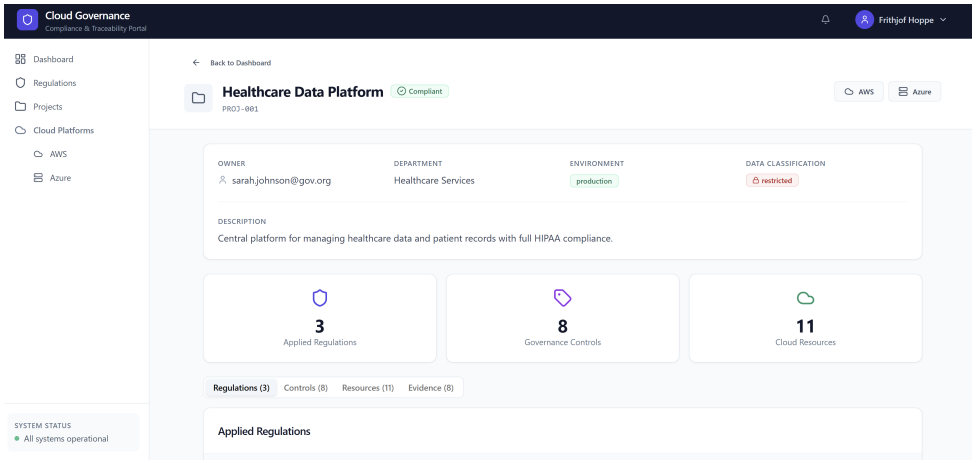


Abbildung 3.7: Detailansicht des Beispielprojekts Healthcare, eigene Abbildung

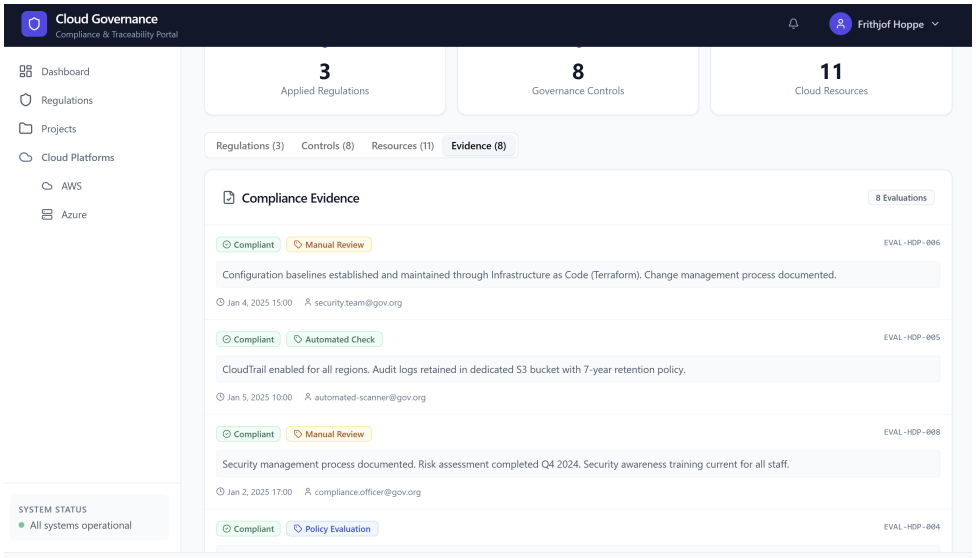


Abbildung 3.8: Detailansicht der beinhalteten Evidence im Beispielprojekt Healthcare, eigene Abbildung

Der PoC* besteht aus den Komponenten in Abbildung 3.9. Das 360°-Cockpit verwaltet mit dem Project Service die Projekte inklusive der zugehörigen Foundations. Der LZ* Provisioning Service provisioniert die Foundations auf den Cloud-Plattformen und hält Referenzen auf die erstellten Cloud Resource Container. Der Hauptanwender des 360°-Cockpits ist der Projektverantwortliche. Das Governance Cockpit verwaltet mit dem Governance Configuration Service die Traceability-Beziehung, auf deren Basis der Traceability Service ein Assessment vornimmt. Der Evidence Service speichert die Artefakte, welche den Stand der Compliance zu einem gewissen Zeitpunkt abbilden. Die Cloud-Administration-Komponenten provisionieren mit dem LZ* Plattform Configuration Service die Grundinfrastruktur der LZ*. Der Provisioning Service / IaC Execution Layer deckt die Interaktionen inkl. Provisionierung auf den Cloud-Plattformen ab. Im PoC* werden lediglich AWS und Azure repräsentativ für weitere grössere Cloud-Plattformen betrachtet.

Um den Fokus auf den genannten Use-Cases zu behalten, wird in Abbildung 3.9 eine Auswahl getroffen, welche Komponenten detailliert implementiert werden. Die grünen Komponenten werden detailliert umgesetzt, um die Funktionalität zu zeigen, die blauen Komponenten werden statisch implementiert, da sie keinen direkten Bezug zur Funktionsweise der Traceability-Architecture aufweisen.

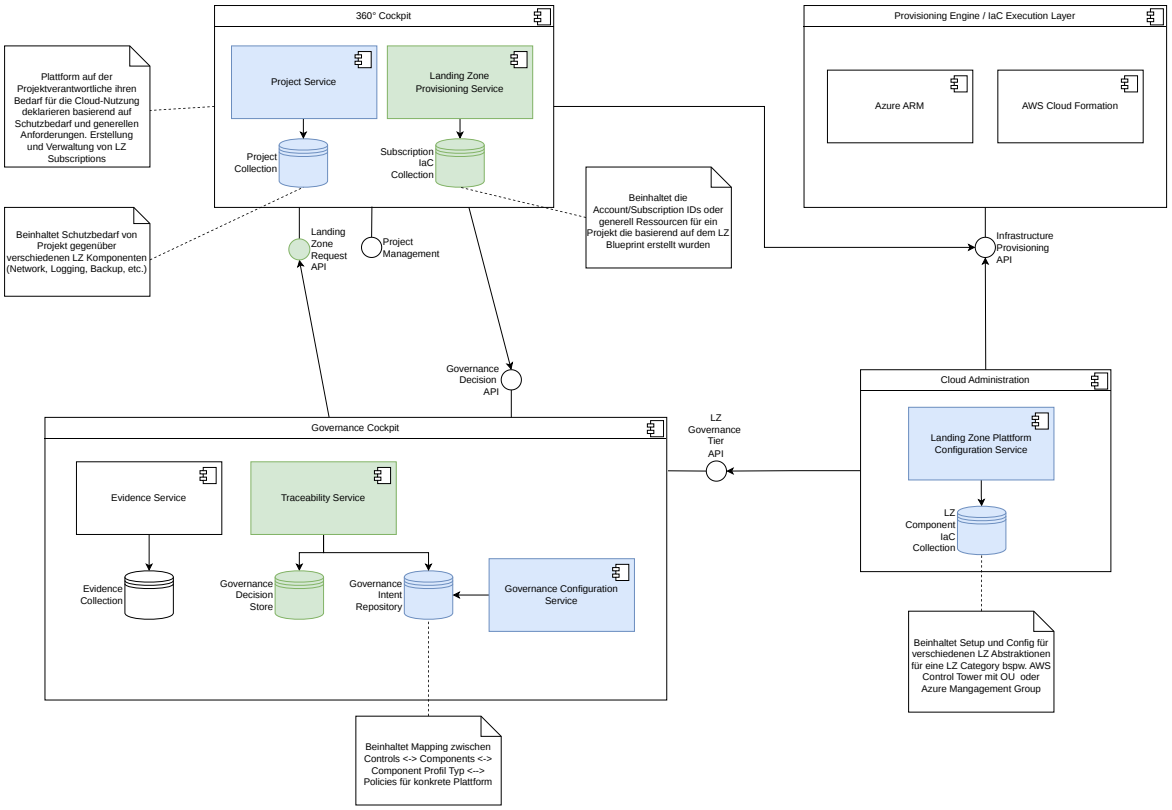


Abbildung 3.9: Komponenten-Diagramm für den PoC*, eigene Abbildung

Im PoC* sind für die Traceability die statisch konfigurierten Inhalte des Governance Configuration Service am relevantesten. Für die implementierte Component-centric Variante werden in Abbildung 3.10 dafür zuerst exemplarische Controls aus der Regulation Si001* gewählt. Die Controls decken absichtlich unterschiedliche regulatorische Gebiete ab, damit eine Zuordnung zu verschiedenen Components, in diesem Fall Storage, Network und Compute, möglich ist. Nach den Controls werden bereits existierende Policies für die Cloud-Plattformen (Azure und AWS) mittels öffentlicher Dokumentation gesucht und wiederum Components/Component-Profiles zugewiesen. Die Component-Profiles werden im PoC* nicht genauer spezifiziert und dienen lediglich als Veranschaulichung. Für eine produktive Anwendung müsste an dieser Stelle ein Security-Intent für ein Profil definiert und basierend auf diesem entsprechende Policies mit Parametern gemapped werden.

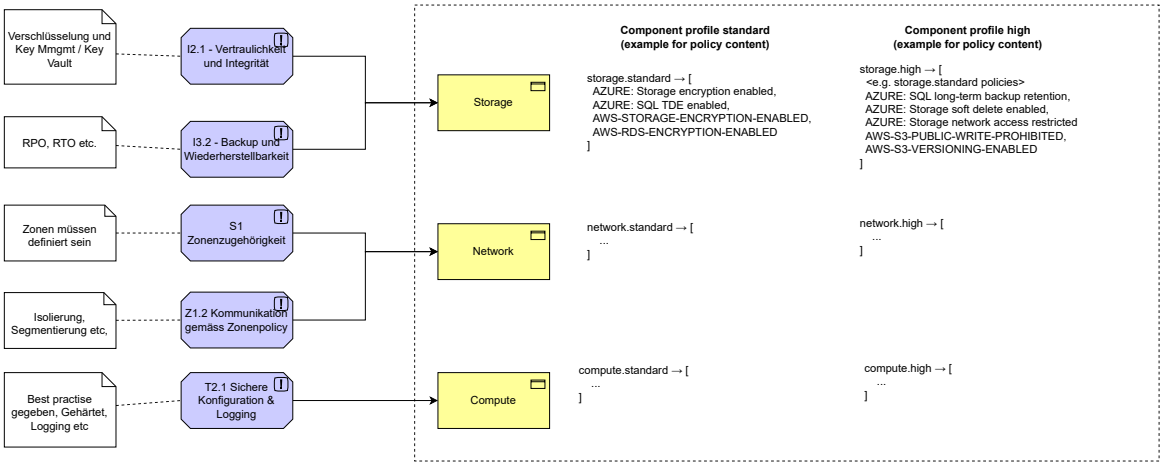


Abbildung 3.10: Statische Traceability-Relationship Komponenten für den PoC*, eigene Abbildung

4 Landing Zone Architektur

Eine LZ* stellt das konzeptionelle Grundgerüst dar, in welchem der Workload für ein Projekt existiert (siehe Unterabschnitt 3.2 (Recherche von Landing Zone/-Komponenten bei Cloud-Plattformen)). Der Begriff Foundation ergibt sich aus den Use-Cases, welche den Scope der Arbeit setzen und sich auf das BIT* beziehen.

Die LZ* bildet trotz der schlussendlichen technischen Implementation bei distanzierter Betrachtung verschiedene Themengebiete ab. Der Nutzen der LZ* besteht darin, diese Gebiete so zu modellieren, dass sie dem Interesse des Unternehmens/der Organisation entsprechen. Hiermit ist gemeint, dass bspw. technische, strukturelle sowie regulatorische Vorgaben der Organisation in Bezug auf den Workload angewendet werden. Die Anwendung passiert hierbei implizit oder explizit durch die Platzierung des Workloads in der LZ* bzw. Foundation. Die Themengebiete, welche für die LZ* relevant sind, lassen sich aus Sicht der EA* auf technische Capabilities zurückführen (siehe Unterabschnitt 3.4 (Erhebung von Landing-Zone-Fähigkeiten)). Die technischen Capabilities stellen somit das technische *Was* dar, welches eine Organisation in Bezug auf Workloads können will. Die LZ* auf der Gegenseite stellt das *Wie* bzw. einen Teil der Implementation dar. Diese verschiedenen Themengebiete, aus welchen die technischen Capabilities resultieren, werden nachfolgend als LZ*-Component bzw. Component bezeichnet. Bei einer vollständigen Implementierung sollten somit theoretisch alle Capabilities in mindestens einer LZ*-Component vertreten sein. Repräsentativ für die diversen identifizierten High-Level-/Level-1-Capabilities in Abbildung 4.1 wird für die Arbeit, passend zum Use-Case (siehe Abbildung 3.5), eine Auswahl vorgenommen. Die Use-Cases (siehe Abbildung 3.5) basieren auf der Fähigkeit der Traceability, welche in Unterabschnitt 4.1 eingehender beschrieben wird.

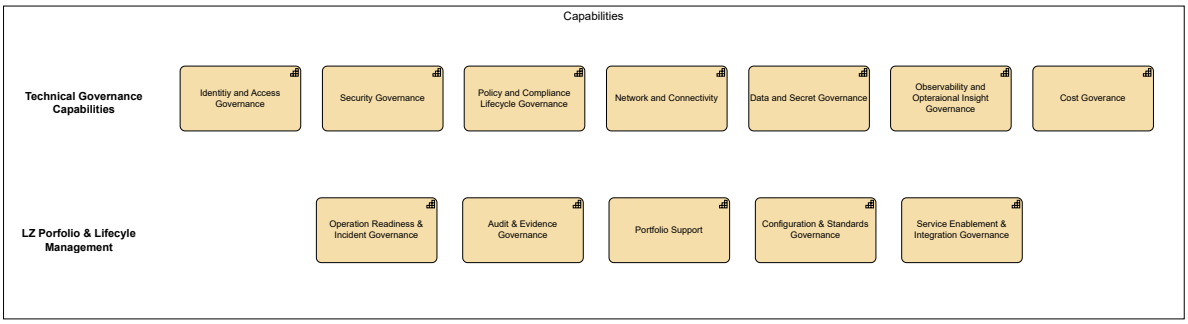


Abbildung 4.1: Reduzierte technische Capability Map mit Level 1 Capabilities. Vollständige Map in Abbildung 3.3, eigene Abbildung

Diese Capability der Traceability hängt in der Capability Map an den Level-1-Capabilities Security Governance, Policy and Compliance Lifecycle Governance, Audit and Evidence Governance sowie Configuration and Standards Governance. Mittels der Traceability wird exemplarisch eine allgemeine LZ*-Architektur in Abbildung 4.2 aufgebaut. Zentraler Angelpunkt in dieser Architektur sind die LZ* Services. Diese Services bilden generelle Blöcke an Ressourcen ab. Denkbar sind hier beispielsweise Storage-Services, Identity-Services etc., welche sich an den Level-1/2-Capabilities orientieren. Aufbauend auf den LZ* Services werden die LZ* Components und die Shared Solution Components abgeleitet. Shared Solutions sind zentrale Components, welche nicht an den Lifecycle von Foundations gebunden sind. Beispiele können hier IAM*-, Audit- oder Site-to-Site-Services sein. LZ*-Components hingegen orientieren sich am Lifecycle einer Foundation. Sie orientieren sich ebenfalls an den Level-1/2-Capabilities, sind aber nur spezifisch für den Scope einer Foundation gedacht. Basierend auf dem IAM*-Beispiel kann als Gegenstück zu einer Shared-Solution-IAM*-Component eine Foundation-spezifische Component existieren,

welche die Integration sicherstellt. Die Plattform-Component am Ende der Struktur stellt als Technology Node eine plattformspezifische Komponente dar, mit welcher die Technology-Services auf den Cloud-Plattformen umgesetzt werden. Für die Anwendung des LZ*-Konstrukts wird ein LZ* Blueprint verwendet. Diese Blueprints stellen Artefakte dar, welche auf den LZ* Components aufbauen und im Sinne von Infrastructure as Code (IaC) provisioniert werden können.

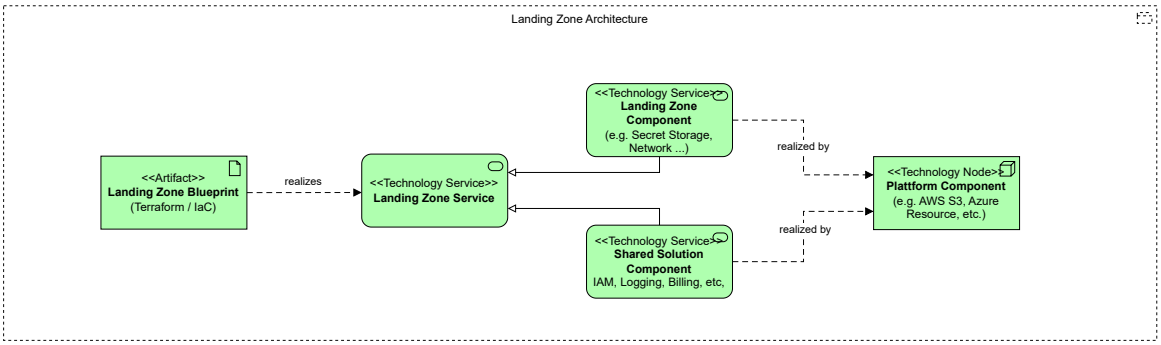


Abbildung 4.2: Vereinfachte LZ* Architektur, eigene Abbildung

4.1 Traceability Architektur

Die Traceability-Architektur stellt eine mögliche Implementation dar, wie Capabilities realisiert werden können. Die Traceability-Architektur ist somit eine Teilarchitektur im grösseren EA*-Kontext und beeinflusst die LZ*-Architektur, wie in Abbildung 4.3 ersichtlich ist. Traceability wird hierbei als Fähigkeit verstanden, den Weg von Anforderungen aus einer Regulation wie Si001* bis hin zu einer Cloud-Ressource nachvollziehen zu können. Dies umfasst sowohl die Aussage, dass ein Objekt wie eine Control aus der Regulation für die Cloud-Ressource relevant ist, als auch den Compliance-Status. Der Compliance-Status sagt wiederum aus, ob die Anforderungen aus der Control gegenüber der Cloud-Ressource erfüllt sind. Summiert man dies über alle Controls für eine Foundation auf, kann somit mittels der Traceability-Architektur eine Aussage über die Compliance von Foundations oder Projekten als Ganzes gemacht werden. Die Traceability soll somit folgende Probleme lösen:

- Wie weist man die relevanten Controls aus einer Regulation zu einer Foundation zu? Der Workload von Projekten in der Cloud muss Vorgaben einhalten, entsprechend muss ein Bezug zwischen Anforderung und Workload hergestellt werden können.
- Wie kann ein Audit für eine Foundation erstellt werden? Eine Organisation wie das BIT* muss eine Aussage darüber machen können, ob der Workload von Projekten konform läuft.

Die Kernidee der Traceability-Architektur ist es, diese Probleme mittels einer Traceability-Beziehung/-Relation zu lösen. Als Relation wird jene zwischen den Controls aus einer Regulation und den Cloud-Ressourcen bezeichnet, welche in einer Foundation laufen. Diese Beziehung soll explizit modelliert werden, um transparent nachvollziehen zu können, welche Controls angewendet werden. Diese Beziehung kann dann verwendet werden, um unter anderem Audits durchführen zu können.

Die zentralen Artefakte in der Architektur bauen auf einem Basismodell auf, welches als Grundlage für die Capabilities erstellt wurde. Eine Regulation bildet ein Regelwerk wie z.B. Si001* ab, welches die verschiedenen Controls enthält. Eine Control ist jeweils eine Anforderung, welche eingehalten werden muss, damit die Control als eingehalten gilt. Eine Control kann durch eine oder mehrere Policies implementiert werden, welche die konkrete Umsetzung der Control darstellen. Diese Policies können entweder selbst definiert oder von den Cloud-Providern bereitgestellt werden.

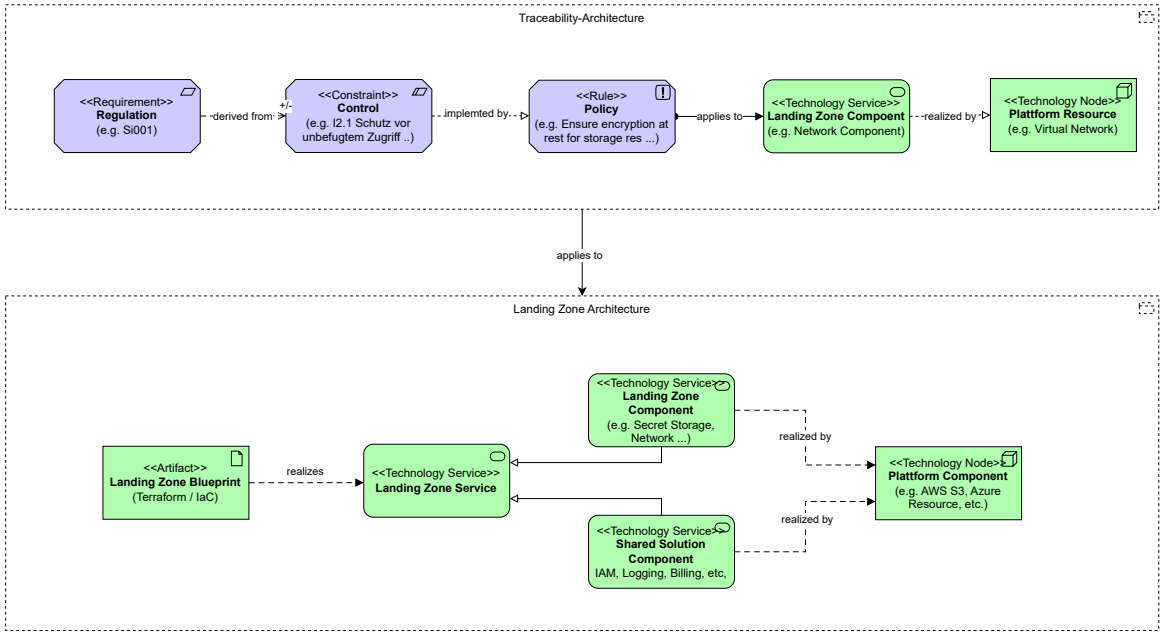


Abbildung 4.3: Traceability-Architektur mit dem Flow der Traceability und den Bezug zur LZ*-Architektur, eigene Abbildung

Die formulierte Idee der Traceability-Architektur auf EA*-Ebene kann in der Ebene der Lösungsarchitektur unterschiedlich umgesetzt werden. Um verschiedene Aspekte der Traceability besser beschreiben zu können, wird deshalb im Rahmen eines PoC* ein Beispiel für eine Lösungsarchitektur gegeben (siehe Unterabschnitt 3.6 (Proof-of-Concept zur Veranschaulichung)). Diese Architektur wird nachfolgend sowie in den verschiedenen Varianten/Ansätzen der Lösungsarchitektur verwendet.

Für eine Prüfung der Compliance eines Workloads ist initial ein Deployment einer Foundation erforderlich (siehe UC6). Dazu werden die Anforderungen an die Governance der Foundation durch den Projektverantwortlichen generell formuliert (siehe Abbildung 4.4). Basierend auf diesen Anforderungen wird pro Komponente in der Lösungsarchitektur eine Placement-Decision getroffen. Als Placement wird hierbei eine Beurteilung der Compliance verstanden. Diese Beurteilung enthält unter anderem, welche Teile einer Regulation relevant sind. Schlussendlich wird die Decision persistiert und basierend auf ihr eine Foundation provisioniert, welche den Workload enthält.

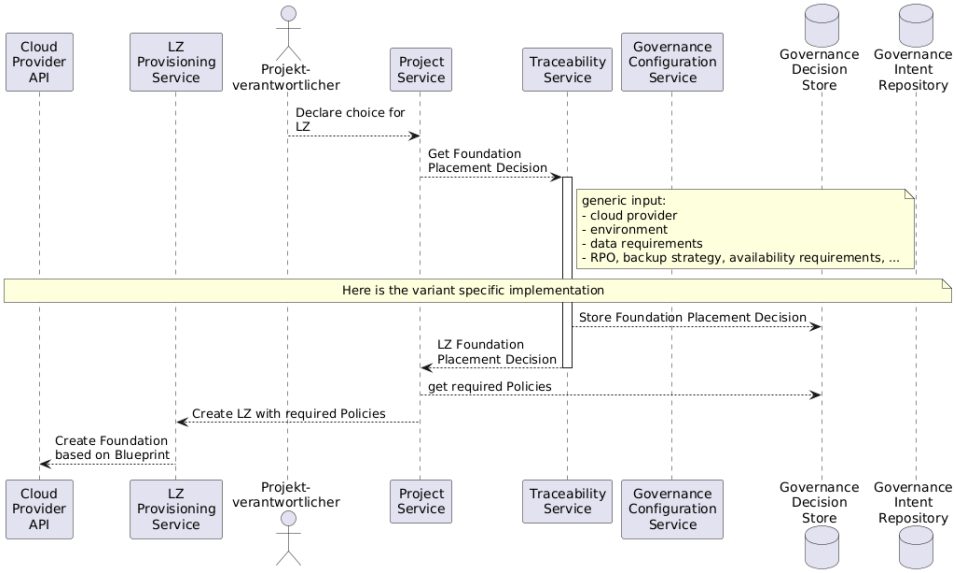


Abbildung 4.4: Grundidee der Zuweisung von Regulation/Controls zu Foundations basierend auf UC6, eigene Abbildung

Nach erfolgreichem Bereitstellen der Foundation kann die Compliance des Workloads innerhalb dieser überprüft werden (siehe Abbildung 4.5). Hierfür wird mittels der Decision und der jeweiligen Traceability-Beziehung eine Rekonstruktion vorgenommen. Basierend auf der Beziehung erfolgt anschliessend die Prüfung der Compliance der individuellen Cloud-Ressourcen. Schlussendlich kann somit die Evidence generiert werden, welche für ein Audit benötigt wird.

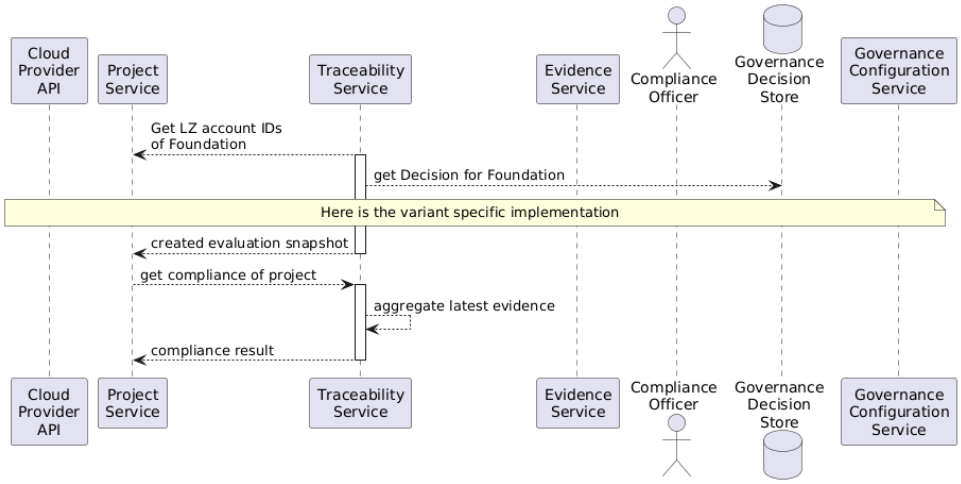


Abbildung 4.5: Grundidee der Prüfung der Compliance einer Foundations basierend auf UC6, eigene Abbildung

Die Darstellung der Traceability-Beziehung basierend auf den Artefakten der Traceability-Architektur (siehe Abbildung 4.3) kann unterschiedlich implementiert werden. Die Varianten, mit denen die Beziehung konkret modelliert und die aufgezeigten Use-Cases implementiert werden können, werden in den nachfolgenden Unterkapiteln gezeigt. Jede Variante umfasst die Grundarchitektur der Traceability-Architecture, ergänzt mit den Objekten für die jeweilige Traceability-Relationship-Variante. Zudem sind die detaillierten Sequenzdiagramme für das Deployment und die Verification enthalten, um die Funktionsweise aufzuzeigen.

4.2 Variante 1: Control-centric Bundling

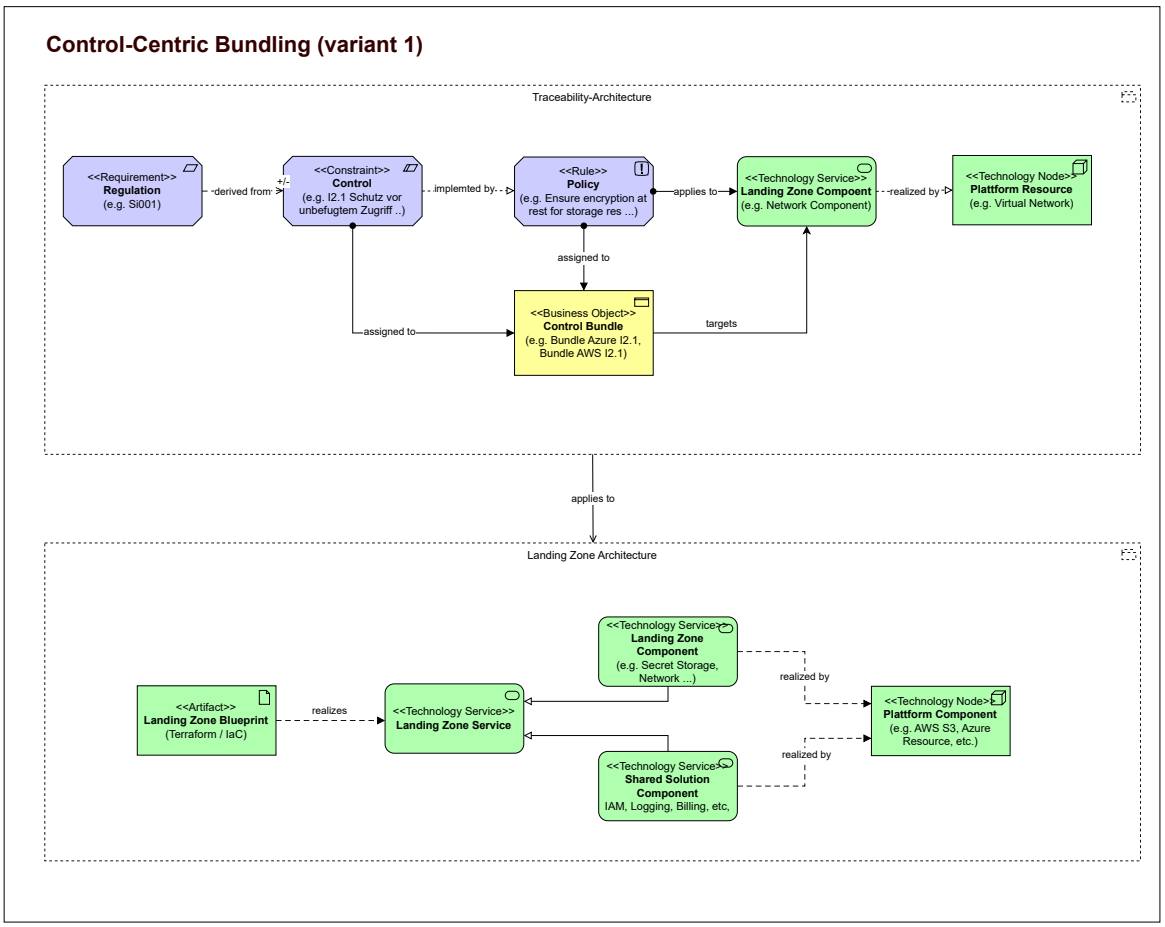


Abbildung 4.6: Control-centric Traceability Architektur Variante, eigene Abbildung

Ansatz: Für jede Control in der jeweiligen Regulation wird ein Mapping zu einem Plattform-Bundle gemacht.

Fokus: Traceability, da die Controls direkt mit den Bundles verknüpft sind.

Artefakte: versionierte Bundles, z. B. in Git, Governance Decisions und Evidence als Datenobjekte

Funktionsweise: In dieser Architektur-Variante werden die Controls direkt mit den Plattform-Bundles verknüpft. Das bedeutet, dass für jede Control in der Regulation ein spezifisches Mapping zu einem oder mehreren Bundles besteht, die die entsprechenden Policies enthalten. Beim Deployment einer LZ* wählt das Governance Cockpit die benötigten Controls basierend auf den Anforderungen der LZ* aus (z. B. Schutzbedarf, Compliance-Anforderungen).

Verantwortlichkeiten: Der Projektverantwortliche definiert die Anforderungen für die jeweilige Foundation. Der Compliance Officer ist zuständig für die Regulations und Controls. Er prüft zudem das Mapping der Bundles durch den Cloud-Experten sowie die Zuweisung der Bundles zu den Foundations.

Maintenance: Da jedes Control direkt mit einem Bundle verknüpft ist, erfordert die Pflege und Aktualisierung der Controls eine sorgfältige Verwaltung der Bundles. Änderungen an ei-

nem Control können Auswirkungen auf mehrere Bundles haben, was eine Versionierung und koordinierte Aktualisierung erfordert.

Adaptability: Neue Controls können durch die Erstellung und Verknüpfung neuer Bundles hinzugefügt werden. Dies setzt jedoch einen soliden Lifecycle-Prozess voraus, sowohl auf Seite des IaC* als auch des Cloud-Providers. Die Komplexität der Auswahl der richtigen Bundles für eine LZ* nimmt zu, wenn die Anzahl der Controls und Bundles wächst.

Abgrenzung zu anderen Varianten: Im Vergleich zu Variante 2, welche die Controls in Component-zentrierte Profiles bündelt, ist diese Variante stärker auf die direkte Verknüpfung von Controls mit Bundles fokussiert, was eine klarere Rückverfolgbarkeit ermöglicht. Variante 3 hingegen bündelt die Controls auf einer höheren Ebene, nämlich auf der Ebene von LZ*-Kategorien, was zu einer stärkeren Abstraktion führt und die Rückverfolgbarkeit eher komplex macht.

Traceability-Strength: Die Rückverfolgbarkeit zu einzelnen Controls ist in dieser Variante am stärksten, da die Controls direkt mit den Bundles verknüpft sind. Es ist einfach nachzuvollziehen, welche Bundles welche Controls implementieren, was eine klare Rückverfolgung ermöglicht.

Laufzeitkomplexität: Deploy Foundation: O(n), Verify Compliance: O(n)

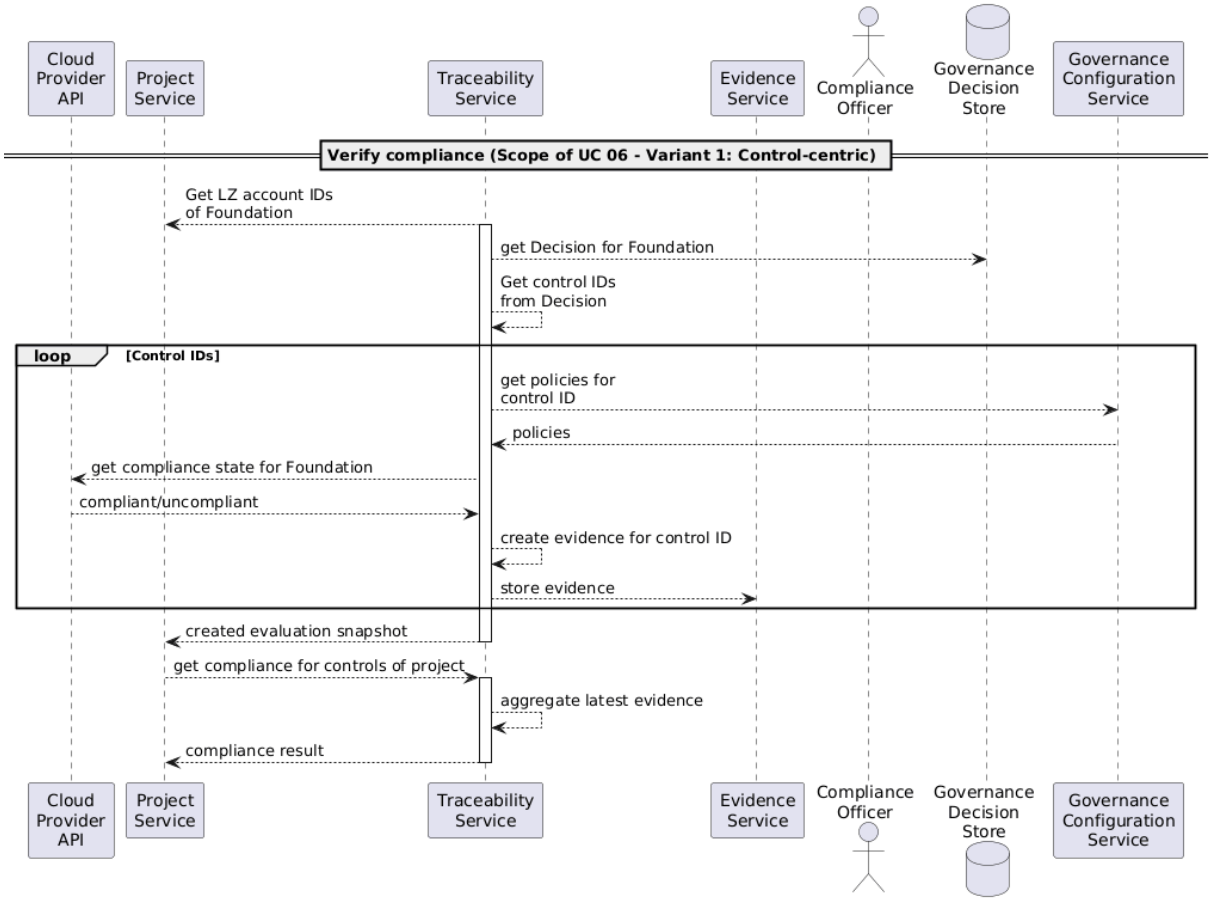


Abbildung 4.7: Sequenzdiagramm für Verifikation der Compliance nach Traceability Architektur Variante 1, eigene Abbildung

Bei der Prüfung der Compliance einer Foundation wird in dieser Variante direkt über die Control IDs iteriert, da die Controls direkt mit den Bundles verknüpft sind. Es ist somit einfach

zu verfolgen, welche Bundles welche Controls implementieren, was eine klare Rückverfolgung ermöglicht.

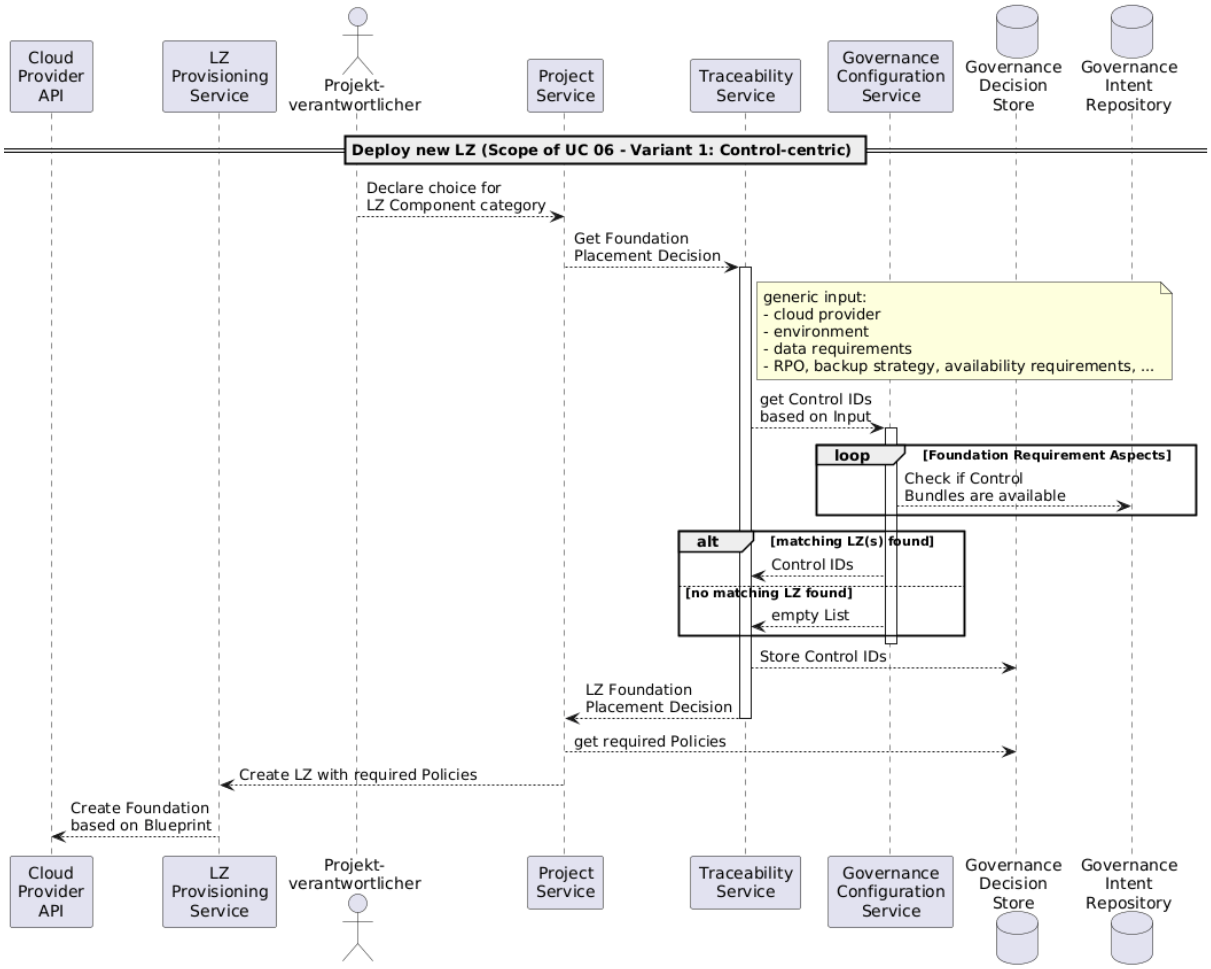


Abbildung 4.8: Sequenzdiagramm für das Deployment einer neuen LZ nach Traceability Architektur Variante 2, eigene Abbildung

Beim Deployment einer neuen LZ* wird in Variante 1 zur Ermittlung der Governance Domain im Intent Repository nach verfügbaren Control-Bundles geschaut, welche die relevanten Controls enthalten. Es wird also direkt nach den Controls geschaut, da diese direkt mit den Bundles verknüpft sind. Es ist somit einfach zu verfolgen, welche Bundles welche Controls implementieren, was eine klare Rückverfolgung ermöglicht.

Fazit: Diese Variante ist diejenige, welche die Traceability-Anforderungen am besten erfüllt, aber auch die aufwändigste in der Umsetzung ist. Sie könnte als «Goldstandard» für die Traceability-Architektur angesehen werden, aber es muss evaluiert werden, ob der Aufwand im Vergleich zu den anderen Varianten gerechtfertigt ist.

4.3 Variante 2: LZ component-centric Bundling

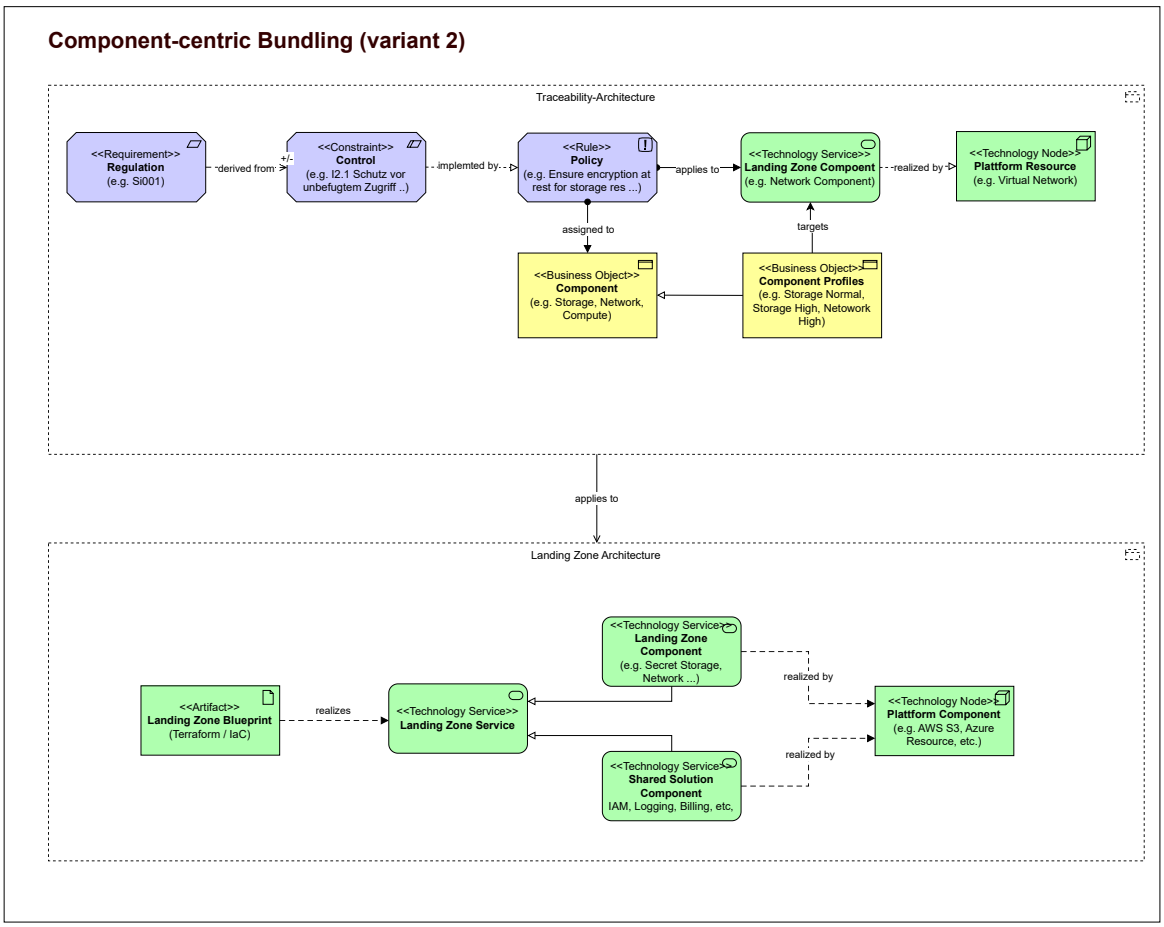


Abbildung 4.9: Component-centric Traceability Architektur Variante, eigene Abbildung

Ansatz: Die Controls werden nach LZ*-Komponenten in Component-Profiles gebündelt.

Fokus: Engineering, Verteilung an Expertengruppen/Teams, aber mit Rückverfolgbarkeit

Artefakte: versionierte Component-Profiles, z. B. in Git, Governance Decisions und Evidence als Datenobjekte

Funktionsweise: Die Controls haben mindestens eine Beziehung zu mindestens einer LZ*-Komponente, welche die jeweilige Control implementiert. Für jede LZ*-Komponente existieren verschiedene Profiles; diese bauen aufeinander auf, das heisst, das nächsthöhere enthält die vorherigen ebenfalls. Die Component Profiles repräsentieren die jeweiligen Abstufungen in den Regulations; z. B. könnte bei Si001* die Sicherheitsstufe «hoch» das Component Profile «storage high» enthalten. Bei mehreren angewendeten Regulations gewinnt jeweils jene mit den höheren Anforderungen. Ein Profile kann mehrere Controls implementieren und bildet das Mapping zwischen Controls und konkreten Policies auf der jeweiligen Zielplattform. Policies können sowohl selbst definiert als auch vom Provider bereitgestellt sein. Der Cloud-Experte ist für das Mapping der Component-Profiles auf die Controls zuständig.

Verantwortlichkeiten: Der Projektverantwortliche definiert die Anforderungen für die jeweilige Foundation. Der Compliance Officer ist zuständig für die Regulations und Controls. Er prüft

zudem das Mapping der Component-Profiles durch den Cloud-Experten sowie die Zuweisung der Component-Profiles zu den Foundations.

Maintenance: In dieser Architektur-Variante werden die Controls in Component Profiles gebündelt; folglich erfordern Änderungen an einer Control nur die Aktualisierung des entsprechenden Profiles, das versionierbar ist. Durch die Versionierbarkeit kann z. B. ein Compliance Officer neue Versionen der Component-Profiles erstellen, ohne dass die bestehenden Foundations direkt betroffen sind. Sobald eine neue Version eines Component-Profiles freigegeben ist, können die entsprechenden Foundations und die übergreifenden Landing-Zone-Components per Provisionierung migriert werden.

Adaptability: Neue Controls können durch die Erstellung und Verknüpfung neuer Versionen von Profiles hinzugefügt werden. Dies ermöglicht eine flexible Anpassung an neue Anforderungen, da die Profiles auf Ebene der Komponenten organisiert sind und Änderungen an einer Control nicht zwangsläufig Auswirkungen auf andere Controls oder Bundles haben.

Abgrenzung zu anderen Varianten: Ähnlich wie in Variante 1 werden hier Controls «gebündelt», jedoch nicht in Control-zentrierten Bundles, sondern in Component-zentrierten Profiles. Die Profiles sind somit eher nach technischen Komponenten organisiert, die von Expertengruppen/Teams implementiert werden, anstatt direkt nach Controls. Variante 3 hingegen bündelt die Controls auf einer noch höheren Ebene, nämlich auf der Ebene von LZ*-Kategorien, was zu einer stärkeren Abstraktion führt.

Traceability-Strength: Die Traceability ist durch die Verknüpfung der Controls mit den Component Profiles gegeben, da jedes Profile die spezifischen Controls enthält, die es implementiert. Allerdings erfordert die Rückverfolgbarkeit zu einzelnen Controls einen zusätzlichen Suchschritt, da die Controls in den Profiles gebündelt sind. Es ist notwendig, ein Manifest zu führen, das auflistet, welche Controls in welchen Component-Profilen enthalten sind, um eine klare Rückverfolgung zu ermöglichen.

Laufzeitkomplexität: Deploy Foundation: O(n), Verify Compliance: O(n)

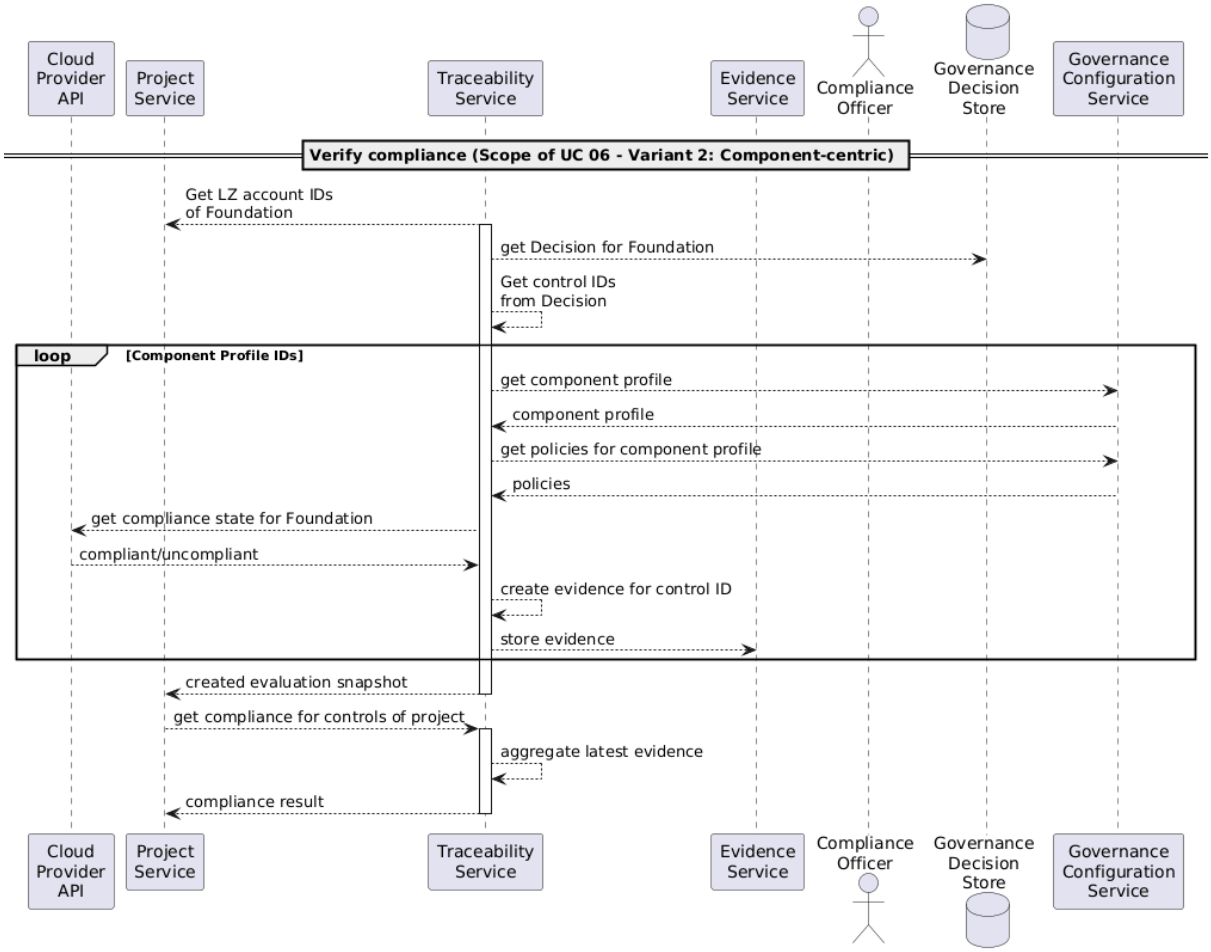


Abbildung 4.10: Sequenzdiagramm für Verifikation der Compliance nach Traceability Architektur Variante 2, eigene Abbildung

In der Loop wird über die Component Profile IDs iteriert, da die Controls in Component-zentrierten Profiles gebündelt sind. Es muss also zuerst das Profil identifiziert werden, welches die relevanten Controls enthält, bevor die Compliance der Foundation überprüft werden kann.

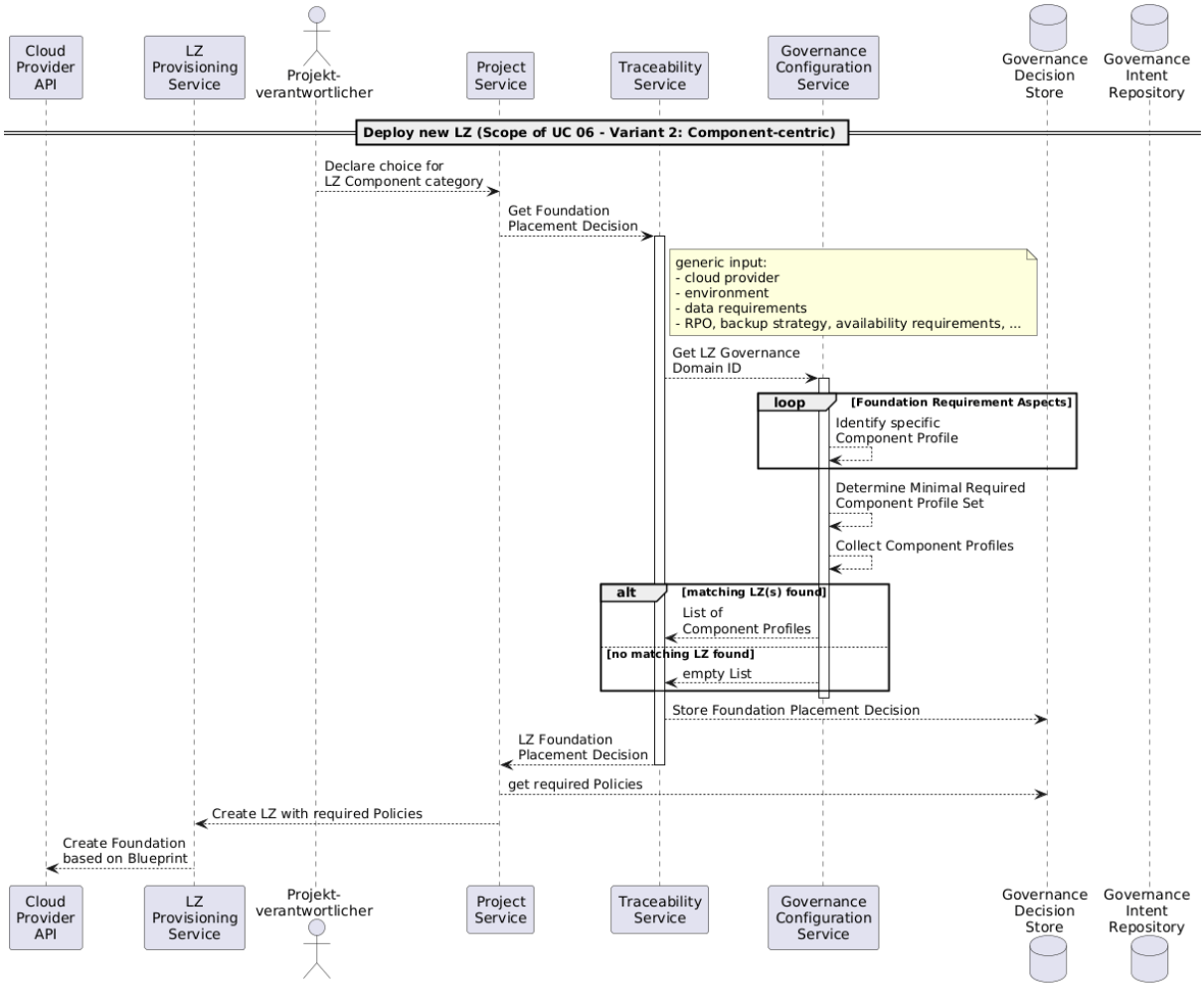


Abbildung 4.11: Sequenzdiagramm für das Deployment einer neuen LZ nach Traceability Architektur Variante 2, eigene Abbildung

Beim Deployment einer neuen LZ* wird in dieser Variante zunächst die Governance Domain der LZ* ermittelt, um daraus die relevanten Component Profiles zu identifizieren. Es wird also nicht direkt nach den Controls geschaut, sondern es wird über die Komponente gebündelt, welche die Controls implementiert. Es muss also zuerst das Profil identifiziert werden, welches die relevanten Controls enthält, bevor die Foundation bereitgestellt werden kann.

Fazit: Diese Variante bietet einen guten Kompromiss zwischen Rückverfolgbarkeit und Praktikabilität, da sie die Implementierungspraxis der Teams berücksichtigt und dennoch eine gewisse Rückverfolgbarkeit ermöglicht. Sie könnte als «Silberstandard» für die Traceability-Architektur angesehen werden, da sie eine praktikable Lösung darstellt, ohne den Aufwand der controlzentrierten Variante zu erfordern.

4.4 Variante 3: LZ category centric Bundling

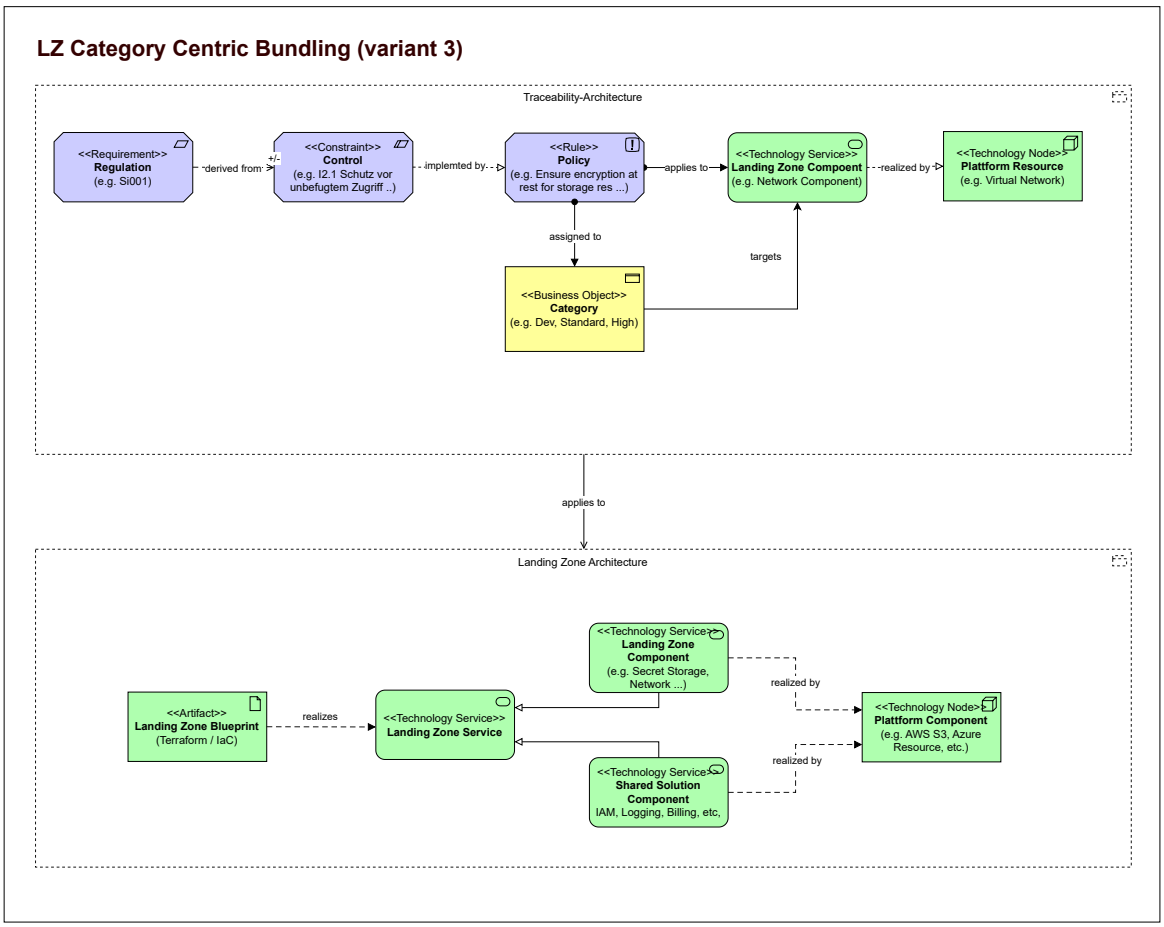


Abbildung 4.12: LZ Category centric Traceability Architektur Variante, eigene Abbildung

Ansatz: Controls werden nach LZ*-Categories gebündelt, welche die Rules für alle Komponenten enthalten. Diese LZ*-Categories umfassen die entsprechenden Policies. Beim Deployen einer LZ* wählt das Governance Cockpit nur die LZ*-Categories aus; der Rest wird abstrahiert.

Fokus: LZ-Kategorien mit Vereinfachung für den Endnutzer, da die Category alles umfasst, ohne dass viel ausgewählt werden muss.

Artefakte: versionierte LZ*-Categories, z. B. in Git, Governance Decisions und Evidence als Datenobjekte

Funktionsweise: Die Controls werden in LZ*-Categories gebündelt, die alle relevanten Regeln für die verschiedenen Komponenten enthalten. Jede Category repräsentiert eine bestimmte LZ*-Kategorie, die eine Risikoklasse oder einen Anwendungsfall abdeckt. Die LZ*-Categories entsprechen den Regulations bzw. ihren jeweiligen Abstufungen. Beim Deployen einer LZ* wählt das Governance Cockpit die entsprechende Category aus, ohne dass sich der Benutzer mit den einzelnen Controls oder Policies auseinandersetzen muss. Diese Herangehensweise bietet eine einfache Anwendung und eine klare Customer Story, da die Kategorien direkt mit den Risikoklassen verknüpft sind. Allerdings führt dies zu einer hohen Duplikation von Controls über verschiedene Kategorien hinweg, und die Rückverfolgbarkeit zu einzelnen Controls erfordert die Pflege eines Manifests, das auflistet, welche Controls in welchen Kategorien enthalten sind.

Verantwortlichkeiten: Der Projektverantwortliche definiert die Anforderungen für die jeweilige Foundation. Der Compliance Officer ist zuständig für die Regulations und Controls. Er prüft zudem das Mapping der LZ*-Categories durch den Cloud-Experten sowie die Zuweisung der LZ*-Categories zu den Foundations.

Maintenance: Änderungen an einer Control erfordern die Aktualisierung aller LZ*-Categories, die diese Control enthalten, was den Wartungsaufwand erhöht. Es ist wichtig, einen klaren Versionierungsprozess zu etablieren, um sicherzustellen, dass Änderungen nachvollziehbar, Versionen referenzierbar und die Historie der Controls und LZ*-Categories gepflegt sind. Dies könnte durch die Verwendung von IaC* in Git erreicht werden, um die Änderungen zu verfolgen und die Versionierung von Controls und Categories zu ermöglichen.

Adaptability: Neue Controls können durch die Erstellung und Verknüpfung neuer Versionen von LZ*-Categories hinzugefügt werden. Allerdings kann dies zu einer erhöhten Komplexität führen, da die Duplikation von Controls über verschiedene Kategorien hinweg die Verwaltung und Aktualisierung der LZ*-Categories erschwert.

Abgrenzung zu anderen Varianten: Im Vergleich zu Variante 1 und 2, welche die Controls in Bundles oder Component-zentrierten Profiles bündeln, bündelt diese Variante die Controls auf einer höheren Ebene, nämlich auf der Ebene von LZ*-Kategorien.

Traceability-Strength: Die Rückverfolgbarkeit zu einzelnen Controls ist in dieser Variante am schwierigsten, da die Controls in den LZ*-Categories gebündelt sind und es eine hohe Duplikation von Controls über verschiedene Kategorien hinweg gibt. Sie erfordert die Pflege eines Manifests, das auflistet, welche Controls in welchen LZ*-Categories enthalten sind, um eine klare Rückverfolgung zu ermöglichen.

Laufzeitkomplexität: Deploy Foundation: O(n), Verify Compliance: O(n)

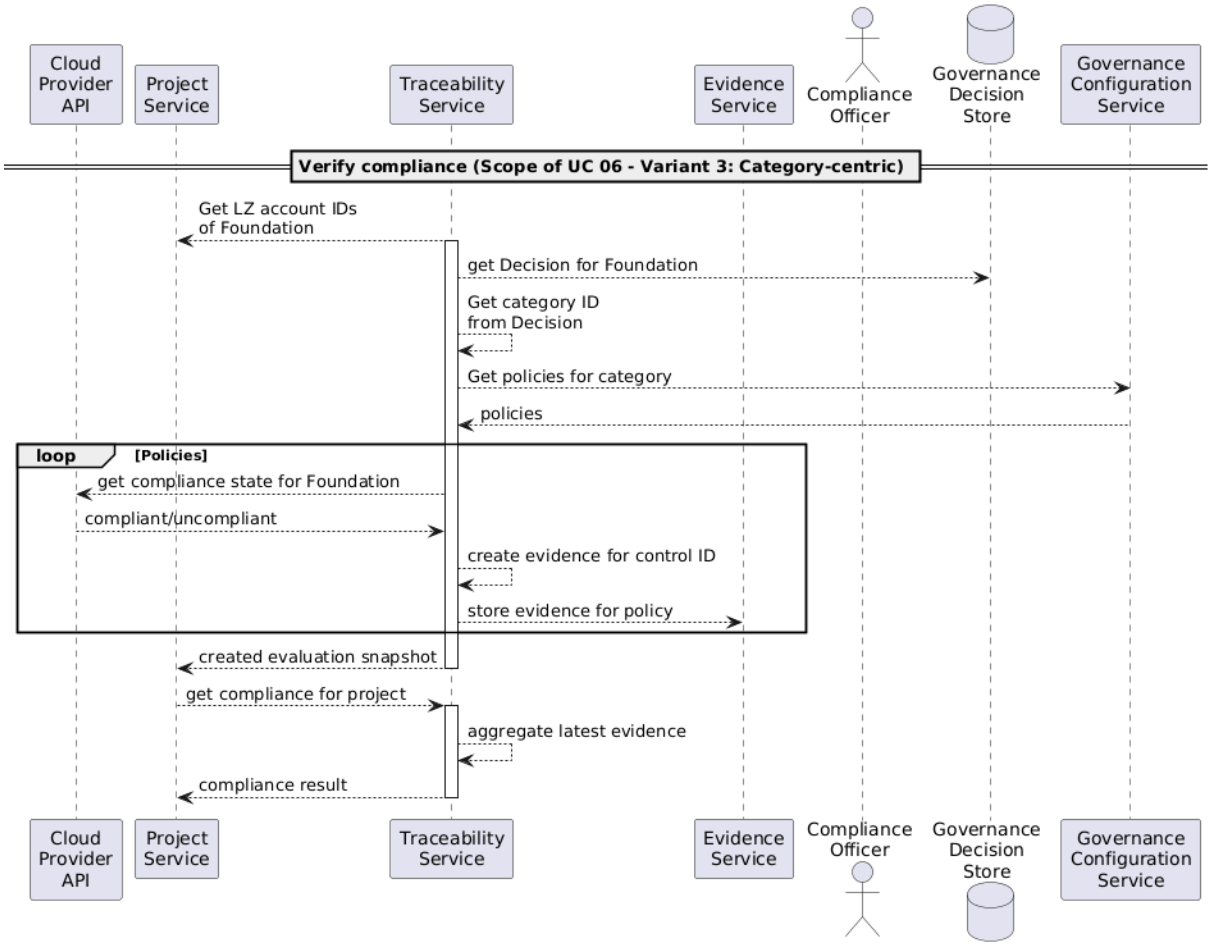


Abbildung 4.13: Sequenzdiagramm für Verifikation der Compliance nach Traceability Architektur Variante 3, eigene Abbildung

In der Loop wird über die jeweiligen Policies iteriert, da die Rückverfolgbarkeit zu einzelnen Controls in dieser Variante nicht direkt gegeben ist. Diese erfolgt nämlich über die Policies, welche die Controls implementieren. Es ist notwendig, die Compliance-Überprüfung auf der Ebene der Policies durchzuführen, um eine Rückverfolgung zu den zugrunde liegenden Controls zu ermöglichen.

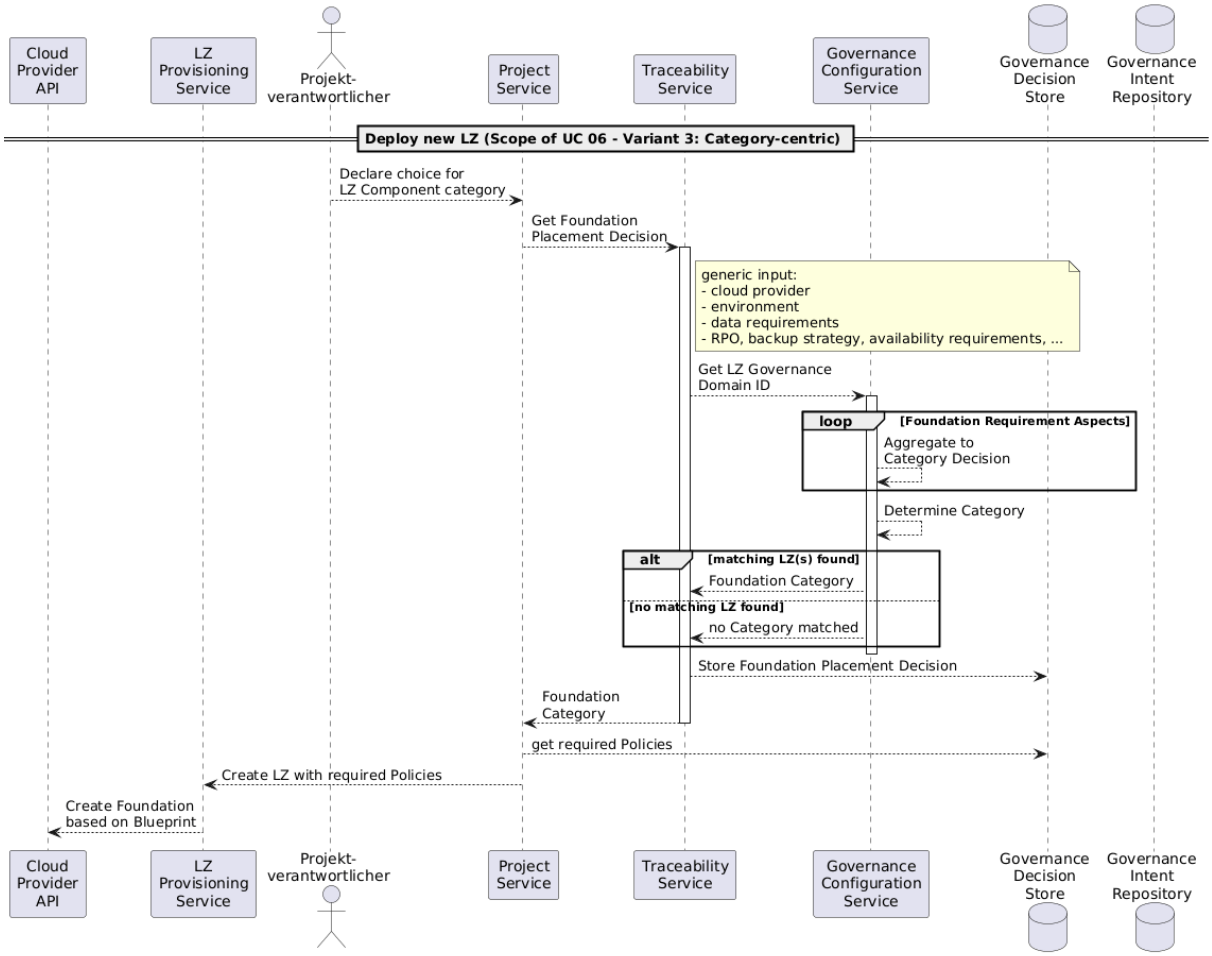


Abbildung 4.14: Sequenzdiagramm für das Deployment einer neuen LZ nach Traceability Architektur Variante 3, eigene Abbildung

Hier wird die Kategorie als Ergebnis der Foundation Placement Decision zurückgegeben, welche dann die Auswahl der LZ*-Categories bestimmt, welche die entsprechenden Controls und Policies enthalten. Die Rückverfolgbarkeit zu den einzelnen Controls erfolgt über die Policies, welche die Controls implementieren, und erfordert die Pflege eines Manifests, das auflistet, welche Controls in welchen LZ*-Categories enthalten sind.

Fazit: Diese Variante bietet die einfachste Anwendung für den Endnutzer, da sie die Komplexität der Auswahl von Controls und Policies abstrahiert. Sie könnte als «Bronze-Standard» für die Traceability-Architektur angesehen werden, da sie eine benutzerfreundliche Lösung darstellt, aber mit erheblichen Nachteilen in Bezug auf Duplikation und Rückverfolgbarkeit einhergeht.

4.5 Variante 4: LZ Provider-native Bundling

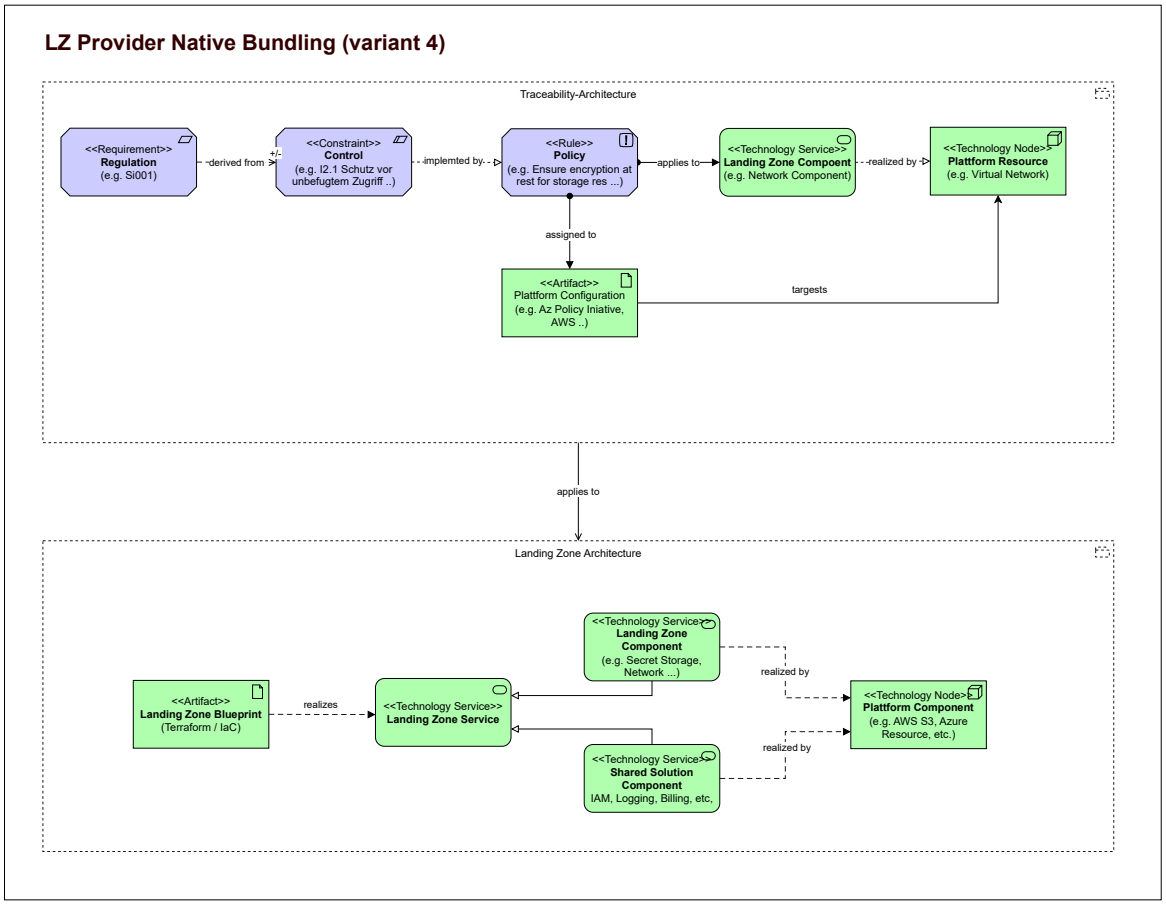


Abbildung 4.15: LZ Provider Native Traceability Architektur Variante, eigene Abbildung

Ansatz: Controls werden direkt durch die native Policy-Abstraction des Providers dargestellt (Azure Initiatives, AWS SCP usw.). Diese werden direkt auf die Komponenten angewendet.

Fokus: Die Anwendung von Richtlinien auf Ressourcen, weniger auf die Nachvollziehbarkeit (à la MVP). Initiatives/Bundles existieren bereits von verschiedenen Cloud-Providern.

Artefakte: versionierte Richtlinien in Git, Governance Decisions und Evidence als Datenobjekte

Funktionsweise: In dieser Architektur-Variante werden die Controls direkt durch die nativen Richtlinienabstraktionen der Cloud-Provider dargestellt. Das bedeutet, dass die Implementierung der Controls direkt auf den Ressourcen erfolgt, ohne dass eine zusätzliche Abstraktionsschicht wie Bundles oder Profiles erforderlich ist. Diese Herangehensweise ermöglicht eine optimale Leistung und eine enge Ausrichtung an den Mechanismen der Provider, da sie die nativen Funktionen und Tools nutzt. Allerdings führt dies zu einer schlechteren Vergleichbarkeit über verschiedene Cloud-Provider hinweg, da jeder Provider unterschiedliche Formate und Strukturen für seine Richtlinien hat.

Verantwortlichkeiten: Der Projektverantwortliche definiert die Anforderungen für die jeweilige Foundation. Der Compliance Officer ist zuständig für die Regulations und Controls. Er prüft zudem die Implementierung der Controls durch den Cloud-Provider bzw. den Cloud-Experten, der die nativen Richtlinien des Providers direkt auf die Ressourcen anwendet.

Maintenance: Da die Controls direkt auf den Ressourcen implementiert werden, erfordert die Wartung und Aktualisierung der Controls eine sorgfältige Verwaltung der Ressourcen und ihrer zugehörigen Richtlinien. Es ist wichtig, einen klaren Versionierungsprozess zu etablieren, um sicherzustellen, dass Änderungen nachvollziehbar, Versionen referenzierbar und die Historie der Controls gepflegt sind. Dies könnte durch die Verwendung von IaC* in Git erreicht werden, um die Änderungen zu verfolgen und die Versionierung von Ressourcen und Richtlinien zu ermöglichen.

Adaptability: Neue Controls können durch die Erstellung und Verknüpfung neuer Ressourcen und Richtlinien hinzugefügt werden. Allerdings kann dies zu einer erhöhten Komplexität führen, da die Verwaltung der Ressourcen und ihrer zugehörigen Richtlinien über verschiedene Cloud-Provider hinweg eine Herausforderung darstellen kann.

Abgrenzung zu anderen Varianten: Im Vergleich zu den anderen Varianten, welche die Controls in Bundles oder Profiles bündeln, bündelt diese Variante die Controls nicht in einer zusätzlichen Abstraktionsschicht, sondern implementiert sie direkt auf den Ressourcen. Dies führt zu einer engeren Ausrichtung an den Mechanismen der Provider, aber auch zu Herausforderungen in Bezug auf Vergleichbarkeit und Rückverfolgbarkeit.

Traceability-Strength: Die Rückverfolgbarkeit zu einzelnen Controls ist in dieser Variante vergleichsweise schlecht, da die Controls direkt auf den Ressourcen implementiert werden und es keine zusätzliche Abstraktionsschicht gibt, die die Controls organisiert. Die Rückverfolgbarkeit zu einzelnen Controls erfordert jedoch eine Normalisierung der Evidence über die verschiedenen Evidenceformate der Provider hinweg, um sicherzustellen, dass die Controls konsistent identifiziert und verfolgt werden können.

Laufzeitkomplexität: Deploy Foundation: O(n), Verify Compliance: O(1) bis O(n)

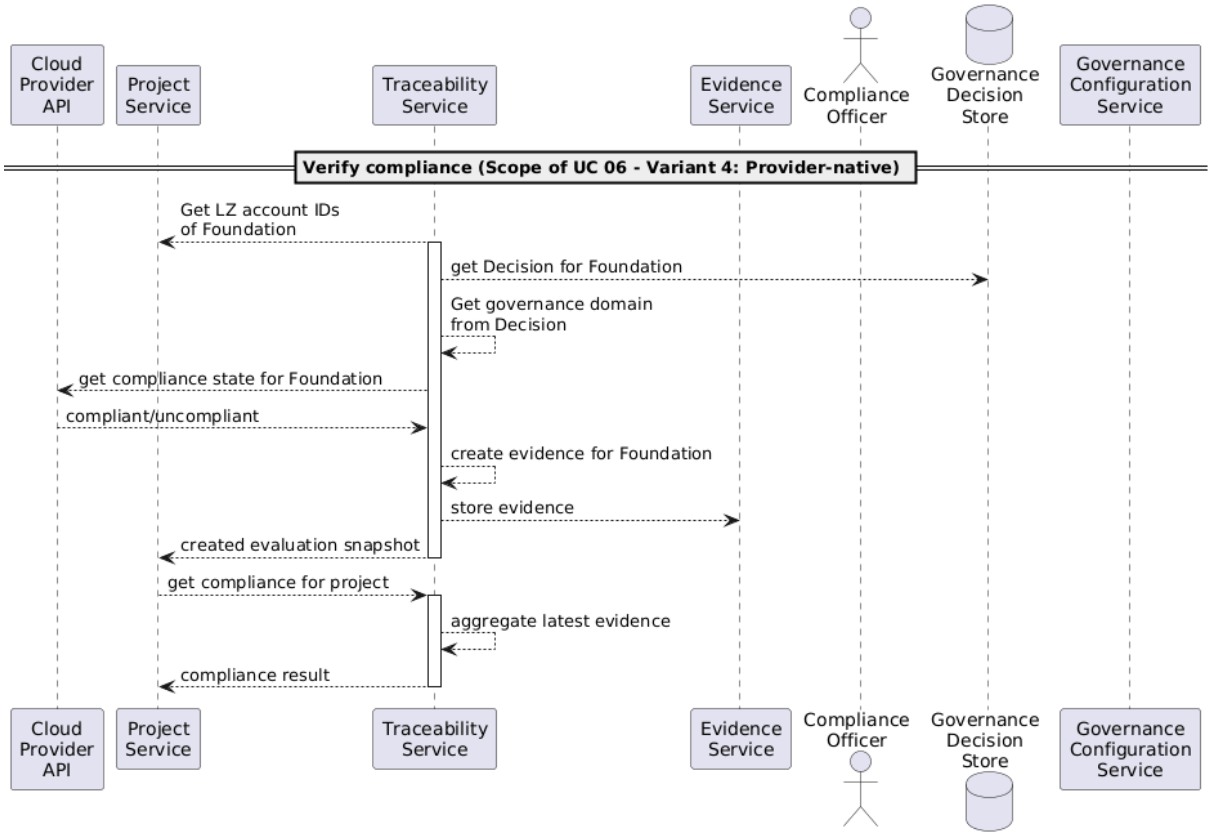


Abbildung 4.16: Sequenzdiagramm für Verifikation der Compliance nach Traceability Architektur Variante 4, eigene Abbildung

In dieser Variante wird die Compliance-Überprüfung direkt über die nativen Mechanismen der Cloud-Provider durchgeführt. Somit kann je nach Provider die Compliance-Überprüfung entweder direkt über die nativen APIs erfolgen, was eine O(1) Komplexität ermöglicht, oder es könnte notwendig sein, über die Ressourcen zu iterieren, um die Compliance zu überprüfen, was eine O(n) Komplexität zur Folge hätte. Die Rückverfolgbarkeit zu den einzelnen Controls erfolgt über die nativen Mechanismen der Provider.

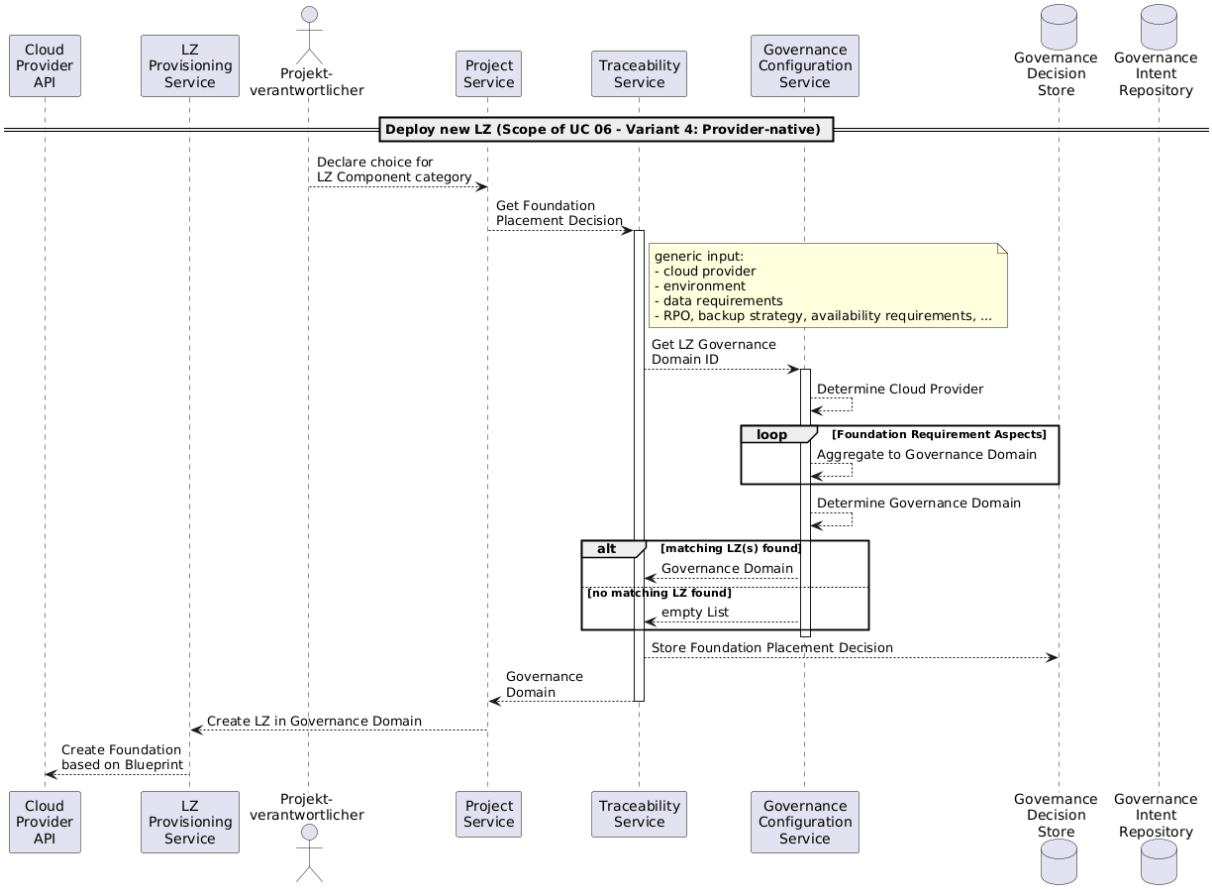


Abbildung 4.17: Sequenzdiagramm für das Deployment einer neuen LZ nach Traceability Architektur Variante 4, eigene Abbildung

Beim Deployment einer neuen LZ* wird in dieser Variante die Governance Domain ermittelt, um die entsprechenden nativen Richtlinien des Cloud-Providers zu identifizieren, welche die Controls implementieren. Es wird also nicht direkt nach den Controls oder Bundles geschaut, sondern es wird über die Governance Domain gebündelt, welche die relevanten Richtlinien enthält.

Fazit: Diese Variante bietet die beste Leistung und Provider-Ausrichtung, da sie die nativen Mechanismen der Cloud-Provider nutzt. Sie könnte als «Performance-Standard» für die Traceability-Architektur angesehen werden, da sie die effizienteste Nutzung der Ressourcen ermöglicht, aber mit erheblichen Herausforderungen in Bezug auf Vergleichbarkeit und Rückverfolgbarkeit einhergeht.

5 Umsetzung des Proof of Concept

Technisch werden das 360°-Cockpit sowie das Governance-Cockpit (siehe Abbildung 3.9) mittels einer Python-Anwendung mit FastAPI implementiert. Die Nutzung des Python-Setups begründet sich damit, dass die CLIs von Anbietern wie Azure und AWS auf Python basieren. Des Weiteren ermöglicht FastAPI mit vergleichbar geringem Aufwand, einen RESTful Service zu erstellen. Um die Trennung der beiden Komponenten 360°-Cockpit und Governance-Cockpit zu betonen, erfolgen gegenseitige Aufrufe jeweils via HTTP-Interface. Zur Persistierung verwenden die Komponenten Project Collection, Evidence Collection, Governance Decision Store und Governance Intent Repository eine MongoDB-Document-Database. Die Wahl einer Document Database begründet sich mit der Abbildung der Traceability-Beziehung als JSON-Objekte, was die Handhabung aus Sicht der Implementierung vereinfacht. Für den Landing-Zone-Plattform-Configuration-Service wird ein statisches IaC*-Setup mit Terraform verwendet. Für AWS stellt dies eine Organization Unit mit zugewiesenen Service Control Policies dar. Für Azure wird eine Management Group mit Policy Assignment verwendet.

Die modellierten Traceability-Beziehungen werden hierbei als statische Inhalte bereitgestellt. Über das Setup hinaus werden weitere Cloud-Ressourcen erzeugt, die in einem funktionalen Setup bereits als gegeben angenommen werden:

- Technische Benutzer/Service-Accounts für die automatisierte wie manuelle Provisionierung von Cloud-Ressourcen
- Verrechnungsstruktur als Voraussetzung zur Erzeugung von Subscriptions/Accounts
- Initiale Strukturen für Komponenten, die über verschiedene Landing-Zones hinweg verwendet werden (z. B. Analytics Workspace, Organization Unit, Management Groups etc.)

Zur Veranschaulichung der technischen Machbarkeit der Traceability-Architektur werden nachfolgend wesentliche Aspekte der Implementierung erläutert. Die Implementierung des PoC* hat sich hierbei auf die Nutzung von Azure und AWS beschränkt, um die Anwendbarkeit auf mehrere Cloud-Plattformen zu zeigen.

5.1 UC4: Import der Traceability-Beziehungen

Im UC4 (siehe Beschrieb UC4) werden die verschiedenen Artefakte erzeugt, die für die Nachvollziehbarkeit der Traceability-Beziehung benötigt werden. Als Grundlage dient das Basismodell für die im PoC* implementierte Variante der Traceability-Architektur Unterabschnitt 4.3 (Variante 2: LZ component-centric Bundling), das in Abbildung 5.1 für die technische Implementierung erweitert wurde. Als Kontext für die Code-Beispiele werden folgende Beziehungen verwendet, die in den nachfolgenden Schritten mittels Code-Beispielen dargestellt werden:

- Die `Component` «Storage» bildet den LZ*-Aspekt von Cloud-Ressourcen ab, die zur Datenspeicherung dienen. Hierbei kann es sich um Datenbanken, Object Storage usw. handeln.
- Die `Regulation` Si001* repräsentiert eines der Regelwerke, die in der BV* angewendet werden. Für die Storage-Component werden hieraus die Abschnitte I2.1 «Vertraulichkeit und Integrität» und I3.2 «Verfügbarkeit» verwendet.
- Es werden die `Component_Profile`s «standard» und «high» definiert, welche als Baselines zu verstehen sind. «high» baut hierbei inhaltlich auf «standard» auf und vererbt jeweils dessen Eigenschaften.
- Den `Component_Profile`s sind verschiedene `Policy`s zugeordnet, die durch die definierten Einschränkungen bzw. Validierungen der Absicht der Controls entsprechen sollen.

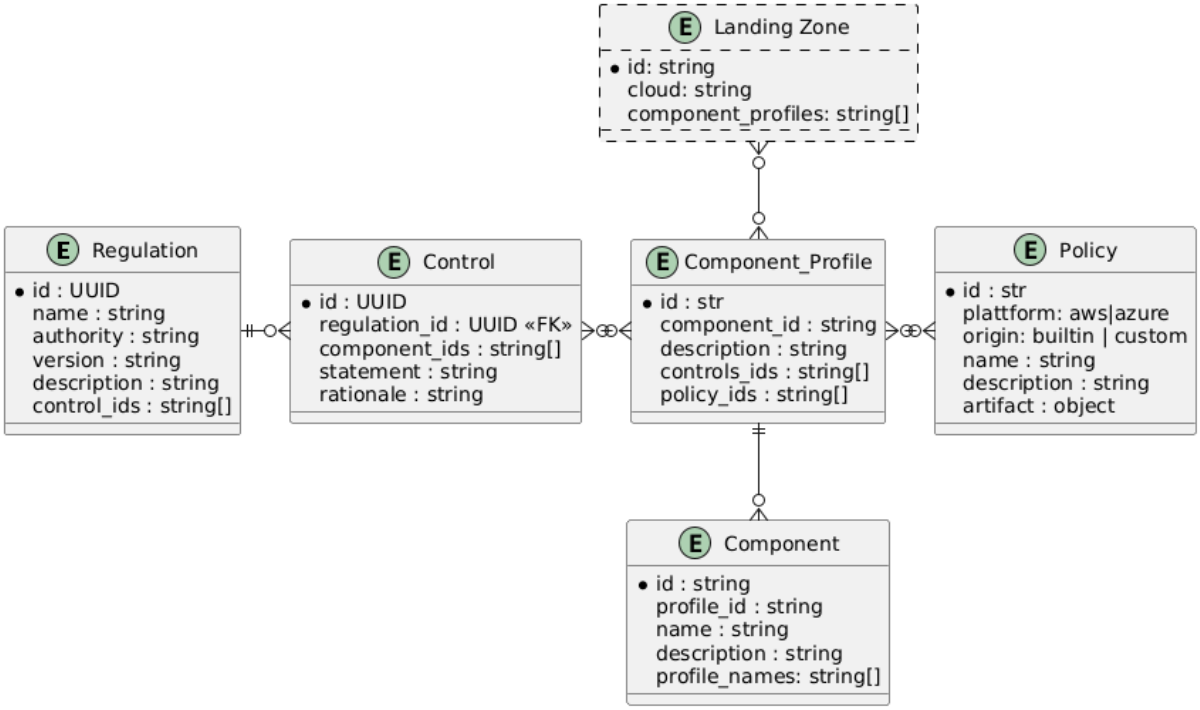


Abbildung 5.1: Datenmodell für Traceability-Beziehung im PoC*, eigene Abbildung

Das `Regulation`-Model bildet zusammen mit den Controls den semantischen Inhalt ab, der gewöhnlich bereits durch die Strukturierung in den verschiedenen Frameworks existiert. Die Zuordnung kann häufig durch ein deterministisches Script erreicht werden. Im PoC* wird dieser Task der Einfachheit halber mittels KI umgesetzt. Die Auswahl der Controls selbst wird manuell ausgeführt und erhebt keinen Anspruch auf Vollständigkeit.

```
{
  "id": "SI001",
  "type": "regulation",
  "name": "Sicherheitsanforderungen IT-Grundschutz (Si001)",
  "authority": "Bundesverwaltung",
  "version": "2022-03-01",
  "description": "Minimale Sicherheitsanforderungen gemaess Si001 fuer IT-Systeme und
    ↪ Anwendungen",
  "control_ids": [
    "SI001-I2.1",
    "SI001-I3.2",
  ]
}
```

Abbildung 5.2: Beispiel für ein Objekt des Typs `Regulation`, dem verschiedene `Controls` zugeordnet sind (eigener Inhalt)

```
{
  "id": "SI001-I2.1",
  "type": "control",
  "rationale": "Schutz vor unbefugtem Zugriff und Manipulation von Informationen.",
  "statement":"Die Vertraulichkeit und Integritaet geschaeftsbezogener Informationen
    ↪ muss durch kryptografische Verfahren entsprechend ihren Schutzanforderungen
    ↪ gewaehrleistet werden.",
  "regulation_ids": ["SI001"],
  "component_ids": ["storage"]
},
{
  "id": "SI001-I3.2",
  "type": "control",
  "rationale":"Schutz vor unbefugtem Zugriff und Manipulation von Informationen.",
  "statement": "Fuer geschaeftskritische Informationen muss eine Backup-Strategie auf
    ↪ Basis des Multi-Generation-Prinzips und Offline-Backups vorhanden sein und
    ↪ umgesetzt werden.",
  "regulation_ids": ["SI001"],
  "component_ids": ["storage"]
```

Abbildung 5.3: Beispiel für Objekte des Typs `Control`, die hier einer `Regulation` und einer `Component` zugeordnet sind (eigener Inhalt)

Basierend auf der Baseline der Component-Profiles werden mittels der Beschreibung aus den Regulationen der ausgewählten Controls I2.1 und I3.2 passende Policies für die unterstützten Cloud-Provider identifiziert und den Profiles je Plattform zugeordnet (siehe Abbildung 5.4). Für eine erste Identifikation wird hierbei KI verwendet, mit einem anschliessenden Abgleich durch manuelle Recherche in den jeweiligen Dokumentationen der Cloud-Provider. Azure bietet mit der zentralisierten Policy-Engine einen direkten Weg, die Korrektheit der Policies mittels `name` oder `policy definition id` zu überprüfen[2]. AWS verfolgt mit den verschiedenen Services, die eigene Policy-Engines besitzen, einen dezentralisierten Ansatz. Entsprechend muss in der Dokumentation der einzelnen Services deren Existenz geprüft werden. Um den unterschiedlichen Ebenen, auf denen die Policies angewendet werden, gerecht zu werden, wird das Attribut `enforcement_layer` auf dem Policy-Model eingeführt, das die Werte `preventive`, `detective` und `runtime` kennt.

[2]Siehe Policy Definitions in Azure Policy Definition Portal

```
{
  "id": "storage.standard",
  "type": "component_profile",
  "component_id": "storage",
  "profile": "standard",
  "description": "Standard-Storage-Profil mit Basis-Datenschutz",
  "control_ids": [
    "SI001-I2.1"
  ],
  "policies": [
    { "platform": "azure", "policy_id": "AZ-STORAGE-ACCOUNT-ENCRYPTION-ENABLED" },
    { "platform": "azure", "policy_id": "AZ-SQL-TDE-ENABLED" },
    { "platform": "azure", "policy_id": "AZ-STORAGE-SECURE-TRANSFER-REQUIRED" },
    { "platform": "azure", "policy_id": "AZ-STORAGE-PUBLIC-ACCESS-RESTRICTED" },
    { "platform": "aws","policy_id": "AWS-STORAGE-ENCRYPTION-ENABLED" },
    { "platform": "aws","policy_id": "AWS-RDS-ENCRYPTION-ENABLED" },
    { "platform": "aws", "policy_id": "AWS-S3-PUBLIC-READ-PROHIBITED" },
    { "platform": "aws", "policy_id": "AWS-S3-PUBLIC-WRITE-PROHIBITED" }
  ]
}
```

Abbildung 5.4: Beispiel für das Storage Component-Profile mit der Baseline Standard und den zugeordneten Policies für Azure und AWS (eigener Inhalt)

Diese Unterschiede beeinflussen auch, wie die einzelnen Policies selbst definiert und zugewiesen werden. Policies können deshalb sowohl Referenzen auf existierende Policies von Providern als auch eigene Definitionen beinhalten. Dies ist insbesondere für AWS erforderlich, da bei einem Assignment häufig eine JSON-Struktur der Zuweisung mitgegeben werden muss.

```
{
  "id": "AWS-STORAGE-ENCRYPTION-ENABLED",
  "type": "policy",
  "platform": "aws",
  "origin": "custom",
  "name": "Storage services must have encryption at rest enabled",
  "description": "Prevents creation of unencrypted storage resources.",
  "artifact": {
    "type": "aws_scp",
    "content": {
      "Version": "2012-10-17",
      "Statement": [
        {
          "Effect": "Deny",
          "Action": [ "ec2:CreateVolume", "rds:CreateDBInstance", "s3:CreateBucket" ],
          "Resource": "*",
          "Condition": {
            "BoolIfExists": { "rds:StorageEncrypted": "false","ec2:Encrypted": "false
    ↪ "}
          }
        }
      ]
    }
  },
  "enforcement_layer": "preventive",
  "scope_hint": "organizational"
}
```

Abbildung 5.5: Beispiel für eine Policy für AWS, welche präventiv auf der Organization Unit direkt zugewiesen wird (eigener Inhalt)

Die Zuweisung von Policies zu den Foundations bzw. LZ*s erfolgt theoretisch über die Component-Profiles bzw. die Policies, die den Component-Profiles zugeordnet sind. Auf welcher Ebene die Zuweisung erfolgt, ist hierbei offen bzw. kann durch die 360°-Cockpit-Komponenten definiert werden. Für den PoC* wird an dieser Stelle eine Vereinfachung vorgenommen, indem für Azure Management-Groups und für AWS Organization Units definiert werden. Diesen providerspezifischen Ressourcen werden bereits vor dem Provisionieren konkreter Foundations Policies zugewiesen. Damit diese Zuweisung möglich ist, werden zwei «Landing-Zones» (siehe Abbildung 5.1) definiert, die ein statisches Set an Component-Profiles besitzen. Die Vereinfachung wird vorgenommen, da eine Provisionierung von Ressourcen in eine Management-Group oder Organization Unit einfacher zu implementieren ist, die Überprüfung der Architektur-Variante jedoch nicht untergräbt.

```
{
  "id": "lz-aws-001",
  "cloud": "aws",
  "component_profiles": {
    "compute": "high",
    "network": "standard",
    "storage": "high"
  }
}
...
{
  "id": "lz-azure-001",
  "cloud": "azure",
  "component_profiles": {
    "compute": "standard",
    "network": "standard",
    "storage": "standard"
  }
}
```

Abbildung 5.6: Beispiel der zur Vereinfachung verwendeten Governance-Domains / LZ* Profiles mit statischen Component-Profiles für Azure und AWS (eigener Inhalt)

5.2 Vorbedingung von UC6: Erstellung der Foundation

Die Evaluierung und die Provisionierung (siehe Vorbedingung Beschrieb UC6) sind in der PoC*-Architektur bewusst zwischen dem 360°-Cockpit und dem Governance-Cockpit getrennt. Das Governance-Cockpit soll vor allem mit den Elementen arbeiten, die Teil der Traceability-Architektur sind, und möglichst nicht mit Spezifika von Cloud-Providern in Berührung kommen. Das 360°-Cockpit hingegen ist für die Ressourcenverwaltung (z. B. innerhalb der Foundations) verantwortlich, die basierend auf den Requests der Projektverantwortlichen erstellt wurden. Entsprechend wird hier auch die Provisionierung verantwortet, und die Spezifika pro Provider sind bekannt.

Als Grundlage für die Zuweisung von Component-Profiles (und schlussendlich Policies) zu einer Foundation wird eine `FoundationDecisionRequestDto` verwendet. Darin formuliert der Projektverantwortliche die Anforderungen, welche die Foundation aus Sicht der Governance erfüllen muss.

```
{
  "name": "myFoundation",
  "environment": "development",
  "data_classification": "public",
  "data_processing": "none",
  "components": {
    "storage": {
      "rpo_minutes": 5000,
      "encryption_at_rest": true,
      "backup_strategy": "none"
    },
    "network": {
      "exposure": "private_only",
      "on_prem_connectivity": "none"
    },
    "compute": {
      "availability": "best_effort"
    }
  }
}
```

Abbildung 5.7: Request zum Evaluieren einer Foundation für eine nicht-produktive Umgebung (eigener Inhalt)

Basierend auf den Parametern wird je nach Variante der Traceability-Architektur ein unterschiedliches Mapping vorgenommen. Für die Component-zentrierte Variante im PoC* ist dies ein Mapping auf Component-Profiles. Zuerst werden hierbei die Profiles für die Anforderungen je Component identifiziert. Die Auswertung erfolgt auf Basis von Thresholds für die verschiedenen Parameter. Exemplarisch wird dies für die Storage-Component gezeigt.

```
STORAGE_PROFILE_LIMITS = {
"standard": {
    "min_rpo_minutes": 1440,   # daily
    "encryption_required": True,
    "allowed_backup_strategies": {
      BackupStrategyDto.NONE,
      BackupStrategyDto.SINGLE
  }
},
"high": {
    "min_rpo_minutes": 60,     # hourly
    "encryption_required": True,
    "allowed_backup_strategies": {
      BackupStrategyDto.SINGLE,
      BackupStrategyDto.MULTI_GENERATION
    }
},
"very_high": {
    "min_rpo_minutes": 15,     # near-real-time
    "encryption_required": True,
    "allowed_backup_strategies": {
      BackupStrategyDto.MULTI_GENERATION
  }
}
}
```

Abbildung 5.8: Simplifizierte Struktur zur Auswertung der Storage Component (eigener Inhalt)

In den Component-Profiles werden die statisch konfigurierten Thresholds aufsteigend mit den Mindestanforderungen der Foundation verglichen. Stimmen diese in allen Werten überein, wurde ein passendes Component-Profile gefunden. Im Falle eines negativen Matchings wird entsprechend ein Leerwert zurückgegeben, der oberhalb abgefangen wird.

```
def _derive_storage_profile(req: StorageRequirementDto) ->
↪  ComponentProfileNameDto:
    for profile in ["standard", "high", "very_high"]:
        limits = STORAGE_PROFILE_LIMITS[profile]

        if req.rpo_minutes is not None and req.rpo_minutes <
        ↪  limits["min_rpo_minutes"]:
            continue

        if req.encryption_at_rest is True and not
        ↪  limits["encryption_required"]:
            continue

        if req.backup_strategy is not None and req.backup_strategy not in
        ↪  limits["allowed_backup_strategies"]:
            continue

        # return the enum member for profile name
        return ComponentProfileNameDto(profile)

    return None
```

Abbildung 5.9: Vereinfachte Identifikation von passendem Component-Profile Level (Source-Code: assessment_service.py #Z495ff) (eigener Inhalt)

Basierend auf dem identifizierten Component-Profile der datenbezogenen Anforderungen wird nachträglich eine Korrektur vorgenommen. Diese soll sicherstellen, dass die Sensitivität der bearbeiteten Daten berücksichtigt wird, da sie einen Einfluss auf alle Components hat. Für den PoC* wird anschliessend die Governance-Domain identifiziert, die den Anforderungen der Foundation genau oder mindestens entspricht. Dies geschieht mittels eines Scorings pro Governance-Domain bzw. LZ*. Bei diesem Scoring wird gemessen, wie gross die Abstände zwischen den für die Foundation identifizierten Component-Profiles und jenen der vordefinierten Governance-Domain sind. Governance-Domains mit zu tiefen Component-Profiles werden bereits vorher verworfen.

```
....
req_compute = effective_compute
req_network = effective_network
req_storage = effective_storage
....
# convert raw documents to typed LandingZone models
lzs: list[LandingZone] = ...

# select eligible landing zones where each component profile >= requested
↪  profile
candidates = []
for lz in lzs:
  # component_profiles may use enum keys/values; normalize to str->str mapping
  cp = lz.component_profiles or {}
  cp_str = {}
  for k, v in cp.items():
    key = k.value
    val = v.value
    cp_str[str(key)] = str(val)

  lz_compute = cp_str.get("compute", "standard")
  lz_network = cp_str.get("network", "standard")
  lz_storage = cp_str.get("storage", "standard")

  try:
    ok = (
        rank.get(lz_compute, 0) >= rank.get(req_compute, 0)
        and rank.get(lz_network, 0) >= rank.get(req_network, 0)
        and rank.get(lz_storage, 0) >= rank.get(req_storage, 0)
    )
  except Exception:
    ok = False

  if ok:
    # score is closeness (lower is better)
    score = (
        (rank[lz_compute] - rank[req_compute])
        + (rank[lz_network] - rank[req_network])
        + (rank[lz_storage] - rank[req_storage])
    )
    candidates.append((score, lz))

if not candidates:
    logger.warning("No landing zone available that meets or exceeds requested
    ↪  component profiles")
    return None

# choose best candidate (minimal score)
candidates.sort(key=lambda x: (x[0], x[1].id))
chosen = candidates[0][1]
...
```

Abbildung 5.10: Identifikation des passenden Cloud Resource Containers im PoC*, assessment_service.py #Z525ff (eigener Inhalt)

Das Assessment des geeigneten Component-Profiles wird schlussendlich zusammen mit dem Placement (für die spezifische Implementierung im PoC*) persistiert und der Foundation zugeordnet. Mittels dieser Zuordnung können zu einem späteren Zeitpunkt während des Deployments die konkret relevanten Policies ermittelt werden.

```
{
  "id": "718fac99-94a4-4e73-840f-23bdb28ba262",
  "foundation_id": "b97fd9c6-43dd-4c68-90c8-8aa77b96825f",

  # General traceability specific part
  "decision": {
    "cloud": "azure",
    "component_profiles": {
      "compute": "standard",
      "network": "standard",
      "storage": "standard"
    }
  },

  # PoC implementation specfic part
  "placement": {
    "type": "landing-zone",
    "id": "lz-azure-001",
    "cloud": "azure"
  }
}
```

Abbildung 5.11: Beispiel einer Governance-Decision für eine Foundation (eigener Inhalt)

Das Deployment der Foundation selbst erfolgt mittels LZ*-Blueprint. Der Blueprint für Azure gliedert sich aus Terraform-Gründen systembedingt in zwei Teile. In Abbildung 5.12 werden die Subscription selbst und die Verknüpfung zur Management-Group provisioniert. Gedanklich entspricht die Subscription an dieser Stelle dem Cloud Resource Container.

```
resource "azurerm_subscription" "foundation" {
  provider          = azurerm.tenant
  subscription_name = var.subscription_name
  billing_scope_id  = var.billing_profile_scope
}

resource "azurerm_management_group_subscription_association" "mgmt_assoc" {
  provider            = azurerm.tenant
  management_group_id = var.management_group_id
  subscription_id     = azurerm_subscription.foundation.id
}

output "subscription_id" {
  value = azurerm_subscription.foundation.subscription_id
  description = "ID of the newly created subscription"
}

output "subscription_name" {
  value = azurerm_subscription.foundation.subscription_name
  description = "Name of the newly created subscription"
}
```

Abbildung 5.12: Base Setup für die Blueprint im PoC* auf Azure (eigener Inhalt)

In Abbildung 5.13 wird anschliessend der effektive Inhalt des Blueprints ersichtlich. Der Einfachheit halber sind dies eine Resource-Group, ein virtuelles Netzwerk und eine Storage Solution.

```
# Content of LZ Blueprint
resource "azurerm_resource_group" "rg_core" {
  name     = "${var.subscription_name}-rg-core"
  location = var.location
}

resource "azurerm_virtual_network" "vnet" {
  name                = "${var.subscription_name}-vnet"
  location            = azurerm_resource_group.rg_core.location
  resource_group_name = azurerm_resource_group.rg_core.name
  address_space       = ["10.10.0.0/16"]
}

resource "azurerm_subnet" "default" {
  name                 = "default"
  resource_group_name  = azurerm_resource_group.rg_core.name
  virtual_network_name = azurerm_virtual_network.vnet.name
  address_prefixes     = ["10.10.1.0/24"]
}

resource "azurerm_storage_account" "storage" {
  name                     = var.storage_account_name
  resource_group_name      = azurerm_resource_group.rg_core.name
  location                 = azurerm_resource_group.rg_core.location
  account_tier             = "Standard"
  account_replication_type = "LRS"
  min_tls_version          = "TLS1_2"
}
```

Abbildung 5.13: Blueprint im PoC* auf Azure (eigener Inhalt)

5.3 UC6: Prüfung der Compliance

Der Stand der Compliance einer Foundation kann geprüft werden (siehe Beschrieb UC6), sobald die Provisionierung durch das 360°-Cockpit stattgefunden hat. Im Rahmen des PoC* wurde lediglich die Compliance-Prüfung auf Azure vollständig umgesetzt. Ausgangspunkt der Prüfung ist die Governance Decision, aus der die Component-Profiles abgelesen werden. Basierend auf den Component-Profiles werden alle Policies zu einer jeweiligen Control-ID aggregiert. Die Auswertung mit Fokus auf die Control begründet sich damit, dass schlussendlich aus allen evaluierten Controls einer Regulation der Compliance-Stand der Regulation abgeleitet werden kann. Der Compliance-Stand einer Control zum Zeitpunkt der Durchführung wird dann als Evidence im Governance Decision Store abgespeichert, wie in Abbildung 5.14 ersichtlich ist. Der Prozess, den Compliance-State zu prüfen, sollte idealerweise in regelmässigen Abständen durchgeführt werden, damit Akteure wie der Compliance Officer ein möglichst aktuelles Lagebild im Governance-Cockpit haben, das auf Evidences aufbaut.

```
evidence_collection = MongoDBCollections.EVIDENCE.value
evidence_ids = []
for control_id in control_ids:
    policy_evidences = []
    for policy_id in control_policies.get(control_id, []):
        policy_doc = await db[policy_collection].find_one({"_id": policy_id})
        ....
        policy = Policy.model_validate(policy_doc)
        ....
        azure_policy_def_id = policy.builtin.policy_definition_id if
        ↪  policy.builtin else None
        ....
        compliance_status_str, description = _get_policy_compliance_status(
            subscription_id, azure_policy_def_id
        )

        # Convert string status to enum
        if compliance_status_str == "COMPLIANT":
            compliance_status = EvidenceComplianceStatus.COMPLIANT
        elif compliance_status_str == "AT_RISK":
            compliance_status = EvidenceComplianceStatus.AT_RISK
        else:
            compliance_status = EvidenceComplianceStatus.NOT_EVALUATED

        policy_evidences.append(
            PolicyEnvidence(
                id=str(uuid4()),
                policy_id=policy_id,
                cloud=CloudPlatform.AZURE,
                compliance_status=compliance_status,
                description=description,
            )
        )
    # Compute overall compliance status from policy evidences
    overall_compliance_status =
    ↪  _compute_evidence_compliance_status(policy_evidences)

    # Set description based on overall status
    if overall_compliance_status == EvidenceComplianceStatus.COMPLIANT:
        description = "All policies compliant"
    elif overall_compliance_status == EvidenceComplianceStatus.AT_RISK:
        description = "One or more policies at risk"
    else:
        description = "Evaluation in progress or incomplete"

    evidence = Evidence(
        id=str(uuid4()),
        control_id=control_id,
        compliance_status=overall_compliance_status,
        ...
    )
    ...
```

Abbildung 5.14: Generierung von Evidences für die Controls des angewendeten Component-Profiles (eigener Inhalt)

Im nächsten Schritt soll gezeigt werden, wie der Evidence-Output, der basierend auf der Traceability-Architektur generiert wurde, dem Compliance Officer im 360°-Cockpit dient, um im Rahmen von UC6 die Compliance prüfen zu können. Hierfür werden zuerst die verschiedenen Ansichten betrachtet, die im Mockup (siehe Appendix A) entworfen wurden:

- Evidence View nach Abbildung A.7: Controls aller Foundations mit individuellem Compliance-State, Zeitpunkt der Prüfung, prüfender Entität und Beschreibung der Control
- Control View nach Abbildung A.5: Controls aller Foundations in einem Projekt, aggregiert mit Compliance-State, Titel der Control und Beschreibung der Control

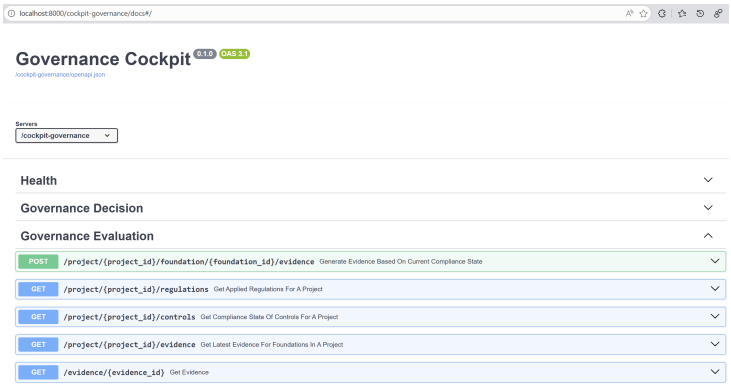


Abbildung 5.15: Endpoints für den Abruf der Compliance States, welche das Mockup theoretisch verwenden könnte, eigene Abbildung

Die Evidence View basiert auf dem `GET /project/{project_id}/evidence`-Endpoint. Dieser Endpoint sucht für alle Foundations, die dem Projekt mit der ID `project_id` zugeordnet sind, den zuletzt verfügbaren Compliance-State pro Control. Diese Auswertung findet, wenn möglich, direkt in MongoDB statt, zur Reduktion der Datenmenge, die zur Laufzeit geladen wird (in Abbildung 5.16 gezeigt). Derselbe Ansatz wurde auch bei den Auswertungen der anderen Endpoints im Tag *Governance Evaluation* angewandt.

```
pipeline = [
    # Match documents for this project
    {"$match": {"project_id": project_id}},
    # Sort by created_at descending (latest first)
    {"$sort": {"created_at": -1}},
    # Group by (foundation_id, control_id), keep first (latest) document
    {
        "$group": {
            "_id": {"foundation_id": "$foundation_id","control_id": "$control_id"},
            "latest": {"$first": "$$ROOT"}
        }
    },
    # Replace root with the latest evidence document
    {"$replaceRoot": {"newRoot": "$latest"}}
]
cursor = db[evidence_collection].aggregate(pipeline)
evidence_docs = await cursor.to_list(length=None)
```

Abbildung 5.16: MongoDB Abfrage für den letzten Compliance State pro Control je Foundation (eigener Inhalt)

Die Response des Endpoints in Abbildung 5.17 sagt aus, dass die Foundation mit der `foundation_id`, beginnend mit `b97fd9c6-...`, den Status `compliant` gegenüber der Control T2.1 aus der Regulation Si001 hat. Der `compliance_status` ergibt sich aus dem Minimalwert der Evaluierung aller betroffenen Policies, die für die Control evaluiert werden. Dies bedeutet, dass der Gesamtstatus entsprechend auch noncompliant bzw. `at_risk` ist, wenn mindestens eine Policy zum gegebenen Zeitpunkt nicht erfüllt wurde.


```
[
  ...
  {
    "id": "b5c0ea60-8375-4886-bc41-f07c5230fd06",
    "control_id": "SI001-T2.1",
    "compliance_status": "compliant",
    "cloud": "azure",
    "description": "All policies compliant",
    "performed_by": "system",
    "project_id": "e72fb364-e689-49f6-96bc-9c66d74ff714",
    "foundation_name": "ref",
    "foundation_id": "b97fd9c6-43dd-4c68-90c8-8aa77b96825f"
  },
  ...
]
```


Abbildung 5.17: Evidence für eine Foundation (eigener Inhalt)

Im Falle einer negativen Evaluierung können Details zu einer Evidence über den Endpoint `GET /project/{project_id}/evidence/{evidence_id}` bezogen werden. Darin ist der Compliance-State für alle Policies aufgelistet, die evaluiert wurden. Somit lässt sich der summierte Compliance-State besser nachvollziehen, und man kann mit bestimmten Policy-IDs beim entsprechenden Cloud-Provider nachforschen.

```
{
  "id": "b5c0ea60-8375-4886-bc41-f07c5230fd06",
  "control_id": "SI001-T2.1",
  "compliance_status": "at_risk",
  "cloud": "azure",
  "deployment_resource": {
    "subscription_id": "bb749eb1-38d3-4e6d-ad5a-f42258bd0e12",
    "region": "switzerlandnorth",
    "deployed_at": "2026-02-22T13:17:38.647000"
  },
  "description": "All policies compliant",
  "performed_by": "system",
  "policy_evidences": [
    {
      "policy_id": "AZ-AKS-DIAGNOSTIC-SETTINGS-TO-LA",
      "cloud": "azure",
      "compliance_status": "at_risk",
      "description": "Not compliant due to reason xyz"
    },
    {
      "policy_id": "AZ-AKS-ENTRA-ID-INTEGRATION-ENABLED",
      "cloud": "azure",
      "compliance_status": "compliant",
      "description": "Policy assignemnt found and compliant"
    },
    ....
  ],
  "foundation_id": "b97fd9c6-43dd-4c68-90c8-8aa77b96825f",
  "project_id": "e72fb364-e689-49f6-96bc-9c66d74ff714"
}
```


Abbildung 5.18: Detailansicht einer bestimmten Evidence (eigener Inhalt)

Die Policy, die intern durch `AZ-AKS-DIAGNOSTIC-SETTINGS-TO-LA` identifiziert wird, löst sich in eine konkrete Policy auf Azure auf. Die Aussage der Policy ist, dass Ressourcen einem bestimmten Log Analytics Workspace zugeordnet sein müssen. Des Weiteren geht hervor, dass es sich um eine präventive Policy handelt, die bereits vor dem Deployment der Landing-Zone-Ressourcen angewendet werden kann. Für Azure erfolgt dies im Rahmen des PoC* direkt über ein Assignment an die Management-Group.

```
{
  "id": "AZ-AKS-DIAGNOSTIC-SETTINGS-TO-LA",
  "type": "policy",
  "platform": "azure",
  "origin": "builtin",
  "name": "Deploy - Configure diagnostic settings for Azure Kubernetes Service to Log
    ↪ Analytics workspace",
  "description": "Deploys/ensures diagnostic settings for AKS are configured to send
    ↪ logs to a Log Analytics workspace.",
  "builtin": {
    "policy_definition_id": "/providers/Microsoft.Authorization/policyDefinitions/6
    ↪ c66c325-74c8-42fd-a286-a74b0e2939d8",
    "parameters": {
      "logAnalytics": {
        "value": "324840a1-dd59-4dec-a255-a48f1dae3d1b"
      }
    }
  },
  "enforcement_layer": "preventive",
  "scope_hint": "landing-zone"
}
```


Abbildung 5.19: Detailansicht einer Policy (eigener Inhalt)

Für die Control View werden die Ergebnisse, die bereits für die Evidence View zusammengestellt werden, übernommen und nochmals pro Projekt zusammengefasst. Somit liefert der Endpoint `GET /project/{project_id}/controls` eine Aussage über den Compliance-Stand der jeweiligen Controls über alle Foundations hinweg. Dies bedeutet (analog zur vorherigen Auswertung), dass das gesamte Projekt gegenüber der Control als `at_risk` dargestellt wird, wenn eine Foundation gegenüber einer Control nicht compliant bzw. `at_risk` ist.

```
[
  {
    "id": "SI001-I2.1",
    "statement": "Die Vertraulichkeit und Integritaet von geschaeftsrelevanten
    ↪ Informationen muessen ihrem Schutzbedarf entsprechend mit Hilfe
    ↪ kryptografischer Verfahren geschuetzt sein.",
    "rationale": "Schutz vor unbefugtem Zugriff und Manipulation von Informationen.",
    "latest_evidence": {
      "id": "dd8d26af-6b5f-48af-8776-3692e330f5d6",
      "control_id": "SI001-I2.1",
      "compliance_status": "at_risk",
      "cloud": "azure",
      "description": "One or more policies at risk",
      "performed_by": "system",
      "project_id": "e72fb364-e689-49f6-96bc-9c66d74ff714",
      "foundation_name": "ref",
      "foundation_id": "b97fd9c6-43dd-4c68-90c8-8aa77b96825f"
    }
  },
  {
    "id": "SI001-S1",
    ...
  }
]
```


Abbildung 5.20: Compliance State der Controls in einem Projekt (eigener Inhalt)

6 Evaluation der Traceability Architektur

Die Evaluation der Traceability-Architektur soll aufzeigen, ob zum einen die vorgeschlagene Struktur der Traceability-Architektur selbst (siehe Abbildung 4.3) und zum anderen ihre Anwendung auf die LZ*-Architektur (siehe Abbildung 4.2) den Anforderungen gerecht werden. Die Überlegung hierbei ist, dass die Traceability-Architektur an der LZ*-Architektur anknüpft und diese dadurch transitiv mit evaluiert werden kann. Für die Evaluation der Traceability-Architektur als solcher werden Architekturprinzipien verwendet (siehe Unterunterabschnitt 3.5.1). Da sich die Traceability-Architektur als Mittel auf EA*-Ebene versteht, soll durch die Konformität mit den Architekturprinzipien gezeigt werden, dass diese Architektur den Vorgaben des Kontextes bzw. der Anwendung entspricht. Die Prinzipien basieren teilweise auf dem Kontext des BIT*. Für die verschiedenen Lösungsarchitekturen bzw. Varianten in den Unterabschnitten werden hingegen Qualitätsattribute (siehe Unterunterabschnitt 3.5.2) betrachtet. Qualitätsattribute erlauben eine bessere Vergleichbarkeit zwischen den Varianten.

Governance

Einer der Hauptnutzen der Architektur ist es, den Erfüllungsgrad der regulatorischen Vorgaben zu beurteilen. Konkret bedeutet dies, sagen zu können, ob Controls aus Dokumenten wie dem Si001* auf eine LZ* angewendet wurden oder nicht. Die Traceability-Architektur unterstützt das Prinzip der Compliance with Law and Policies durch die Modellierung einer Traceability-Beziehung zwischen Governance-Artefakten und den technischen Komponenten der LZ*.

Konkret werden Beziehungen zwischen den Controls und den Cloud-Ressourcen erstellt. Dafür werden zuerst die Controls und Policies identifiziert, die sowohl für eine LZ*-Component als auch für eine Shared Solution Component relevant sind. Basierend auf diesem Mapping ist es anschliessend möglich, beim Provisionieren einer LZ* zu bestimmen, welche Controls relevant sind. Mit diesem Schritt wird ebenfalls das Prinzip der Data Governance unterstützt, da die Aussage darüber, welche Art von Daten in der Foundation verarbeitet wird, ebenfalls in das Assessment einfliesst. Durch den Aufbau der Traceability-Beziehung beim Erstellen der Foundation kann letztendlich nachvollzogen werden, ob Controls, die an der Relation hängen, umgesetzt werden.

Abstraction and Plattform Independence

Eine der Hauptvoraussetzungen, die bereits aus den Forschungsfragen hervorgeht, ist die Anwendbarkeit auf einen Multi-Cloud-Kontext. Die Erwartung an die Traceability-Architektur ist somit, dass Komponenten nicht auf plattformspezifischen Ansätzen aufbauen. Die Architektur unterstützt das Architectural Abstraction Principle, indem von der allgemeinen Struktur der Regulation und Controls ausgehend modelliert wird. Diese Struktur findet sich nicht nur beim Si001*, sondern auch bei anderen Frameworks wie bspw. jenen des NIST*. Diese Abstraktion zeigt sich auch in der LZ*-Architektur als solcher, welche mit den LZ*-Services, den Komponenten und Shared-Services den Cloud-Provider entkoppelt.

Structural Architecture

Die Orientierung an den verschiedenen Akteuren aus den Use-Cases ist erforderlich, um eine Ausrichtung an den Abläufen im Anwendungskontext zu gewährleisten. Die Architektur unterstützt das Prinzip der Service-Orientierung, indem die Traceability-Architektur bereits nutzerzentriert gedacht wird. Die Betrachtung der Traceability-Beziehung erfolgt aus der Sicht des Compliance-Officers. Entsprechend sind auch die Informationen, die abgeleitet werden können, Governancezentriert, um Auditierbarkeit zu ermöglichen. Die Nutzung einer einheitlichen Terminologie ist vor allem bei der Einführung einer neuen Architektur wichtig, um unklare Verwendungen von Begriffen zu vermeiden. Sowohl die Traceability- als auch die LZ*-Architektur selbst bauen auf einem gemeinsamen Metamodell auf (siehe Abbildung 3.2) und unterstützen damit das Prinzip

des Reference Model Principles. Um die genannte Auditierbarkeit den Nutzern verfügbar zu machen, ist jedoch auch eine Automatisierung erforderlich. Die Architektur unterstützt das Prinzip der Standardisation Automatization dadurch, dass die Traceability-Beziehung maschinell lesbar ist und ausgewertet werden kann. Hierfür wird zwar kein bestehender Standard genutzt, durch den Aufbau auf dem Metamodell jedoch ein definiertes Vorgehen verwendet.

Security Architecture

Minimale Berechtigungen für die Prüfung der Traceability-Beziehung sind ideal, damit einzelne Akteure keine Administratorprivilegien besitzen müssen, um verschiedene Ressourcen zu prüfen. Die Traceability-Architektur unterstützt das Prinzip des Security by Design, da unpersönliche Managed Identities verwendet werden können, um potenzielle Evidences über den Compliance-Stand generieren zu können. Die Architektur selbst soll einen klaren Security-Intent besitzen, um darauf basierend Security-Models aufbauen zu können. Die Traceability-Architektur unterstützt das Security Reconfigurability Principle dadurch, dass der Scope der Architektur selbst lediglich die Rekonstruktion von Beziehungen zwischen Ressourcen umfasst. Die Anwendung schreibender Operationen ist von der Traceability-Architektur getrennt. Die tatsächliche Durchsetzung von Sicherheitsmechanismen erfolgt weiterhin durch die jeweiligen Cloud-Plattformen selbst. Die Traceability-Architektur ermöglicht somit primär die Nachvollziehbarkeit und Zuordnung von Sicherheitsanforderungen zu konkreten Policies, übernimmt jedoch keine eigenständige Enforcement-Funktion.

6.1 Evaluation Variante 1

Die Evaluation der Control-zentrierten Variante zeigt eine Architektur mit hoher Portability und Auditability bei gleichzeitig geringer struktureller Komplexität. Durch die direkte Zuordnung von Controls zu plattformspezifischen Policies entsteht eine klare und nachvollziehbare Traceability-Struktur, welche insbesondere aus Governance-Sicht eine transparente Abbildung regulatorischer Anforderungen ermöglicht.

Gleichzeitig führt der geringe Abstraktionsgrad dazu, dass Kontextinformationen auf Workload-Ebene nur eingeschränkt berücksichtigt werden können. Dies wirkt sich insbesondere auf die Traceability Strength aus, da zwar überprüft werden kann, ob Policies im Sinne eines Controls erfüllt sind, jedoch nicht zwingend, ob diese Zuordnung dem konkreten Nutzungskontext eines Workloads entspricht. Auch die Performance der Evaluation kann durch wiederholte Prüfungen identischer Policies beeinflusst werden, da keine Bündelung über zusätzliche Strukturierungsebenen erfolgt.

Insgesamt positioniert sich die Variante als Governance-zentrierter Ansatz mit hoher Modellvereinfachung und klarer Auditierbarkeit, der jedoch zugunsten dieser Einfachheit eine geringere Differenzierung der Traceability-Beziehungen in Kauf nimmt. Sie eignet sich insbesondere für Szenarien, in denen regulatorische Transparenz und plattformübergreifende Übertragbarkeit wichtiger sind als eine kontextuelle Feinsteuerung der Compliance-Bewertung.

6.1.1 Bewertung der Quality Attributes

Die Bewertung mittels der Quality-Attribute erzielt einen totalen Score von *3.18* und wird nachfolgend textuell erläutert.

ID	$s(q)$	**Quality-Attribute**
Q1	2	Performance
Q2	4	Portability

ID	$s(q)$	Quality-Attribute
Q3	4	Auditability
Q4	2	Automatability
Q5	3	Maintainability
Q6	3	Adaptability
Q7	4	Complexity
Q8	2	Traceability Strength

Tabelle 6.1: Übersicht Bewertung der Auswahl an Quality-Attributen nach Unterunterabschnitt 3.5.2

Performance

Mit dem geringen Mapping von lediglich den Controls direkt auf die Policies wird der Aufwand für die Auflösung der Traceability-Beziehung reduziert. Die Performance wird jedoch massgeblich durch die Aufrufe auf die externen Cloud-Provider-APIs bestimmt, um den aktuellen Compliance-Status zu erhalten. Relevant ist hierbei, dass potenziell Policies mehrmals verschiedenen Controls zugeordnet werden könnten, da keine Abstraktion existiert, die diese bündelt. Die Evaluation fällt deshalb tiefer aus.

Portability

Regulations und Controls sind naturgemäss bereits plattformunabhängig formuliert. Dies bedeutet, dass das Mapping von Controls zu den Policies auf weitere Cloud-Plattformen übertragen werden kann. Die Portability ist demnach hoch einzustufen.

Auditability

Die Objekte, welche die Traceability-Beziehung abbilden, können Metainformationen enthalten. Dadurch kann nachvollzogen werden, wer z. B. Controls erstellt hat und warum Policies einer Control zugewiesen werden. Durch die geringe Abstraktion, die lediglich aus der Control besteht, ist auch keine Art von Verschmelzung möglich, womit die Audit-Pfade klar bleiben. Die Auditability ist deshalb hoch.

Automatability

Die explizite Modellierung der Traceability-Beziehungen erlaubt eine maschinelle Prüfung, was somit die Automatisierbarkeit erhöht. Gleichzeitig kann jedoch automatisiert keine genaue Aussage darüber gemacht werden, ob alle Policies einer Control für den Workload relevant sind. Denkbar ist hier die Unterscheidung zwischen Production- und Development-Workloads. Die Generierung von Audit-Reports basierend auf erzeugten Evidences ist nicht Teil der Traceability-Architecture. Die Zuordnung von Policies zu Controls erfolgt separat und ist möglicherweise mittels KI automatisierbar, benötigt jedoch eine manuelle Nachprüfung.

Maintainability

Die Modelle/Objekte, welche für die Traceability-Beziehung verwendet werden, können versioniert und individuell weiterentwickelt werden. Gleichzeitig erfordert die Zuordnung von Policies zu Controls eine kontinuierliche Prüfung, z. B. bei Änderungen seitens der Cloud-Provider. Denkbar sind hier Deprecations von Policies, Anpassungen von Parametern etc.

Adaptability

Anpassungen oder Ergänzungen der Controls sind möglich, solange das Mapping zu den Policies erhalten bleibt. Wird das Mapping angepasst, muss entsprechend auch der Algorithmus zur Auflösung der Traceability-Beziehung angepasst werden. Entsprechend ist die Adaptability hoch.

Complexity

Die Modellierung der Beziehung orientiert sich recht stark an der Struktur der Regulation mit den Controls. Dies bedeutet, dass so gut wie keine weitere Abstraktion obendrauf vorhanden ist.

Traceability Strength

Die modellierte Traceability ermöglicht zwar eine Aussage darüber, ob die Policies, die für eine Control relevant sind, erfüllt wurden, aber nicht zwangsläufig, ob dies der Absicht des Workloads gerecht wird. Werden bspw. alle Policies für einen Production-Workload zugewiesen, könnten Workloads für einen Non-Production-Workload negativ evaluiert werden, weil kein Scoping möglich ist.

6.1.2 Trade-Offs

Explicit Trace Modeling vs. Implicit Derivation

Mit der Modellierung der Controls verfolgt die Variante einen expliziten Ansatz, der gleichwohl einen impliziten Charakter besitzt. Implizit, da die Struktur mit Regulation und Controls direkt aus existierenden Vorgaben wie Si001* abgeleitet ist. Explizit, da dem Cloud-Provider diese Struktur nicht zwangsläufig bekannt ist. Ein Mapping der Anforderungen durch den Projektverantwortlichen im 360°-Portal auf die erforderlichen Controls ist komplexer, da wegen der Implizität der Modellierung keine Abstufung möglich ist.

Governance centric Traceability vs. Engineering centric Traceability

Die Control-centric-Variante orientiert sich stark am Akteur des Compliance-Officers, da dieser Akteur regulatorischen Dokumenten am nächsten ist. Mit der Control-basierten Sicht ist die Erstellung von Audits möglicherweise einfacher. Aus Engineering-centric-Sicht fehlt der Variante die vollständige Automatisierbarkeit, da kein Mapping auf den Kontext der Workloads möglich ist. So weiss der Projektverantwortliche zwar, dass bspw. für einen Non-Production-Workload bewusst die Konformität gegenüber gewissen Policies vernachlässigt wurde, der Compliance-Officer sieht dies aber nicht.

Traceability Depth vs. Model Simplicity

Die hohe Model Simplicity ist aus technischer Sicht einfacher, da weniger Mapping erforderlich ist. Gleichzeitig wird durch den reduzierten Einfluss der Requirements des Akteurs Projektverantwortlicher die Traceability Depth reduziert. Reduziert im Sinne dessen, dass, auch wenn das Control-Policy-Mapping klar ist, der Workload-Kontext aus den Requirements (z. B. non-prod/prod) vernachlässigt wird. Dadurch sinkt der Informationsgehalt, den man aus der Evidence erhält.

Central Traceability Model vs. Federated Traceability Models

Das Modell in dieser Variante ist offen bzw. unterliegt denselben Einschränkungen wie die Regulations, auf denen es aufbaut. Sprich: Alle Regulations mit Control-basiertem Aufbau können implementiert werden, selbst wenn das Modell nicht federated ist.

6.2 Evaluation Variante 2

Die Evaluation der Component-spezifischen Variante zeigt insgesamt eine ausgewogene Architektur mit guten Eigenschaften hinsichtlich Portability, Maintainability und Auditability. Die klare Zuordnung von Policies zu Component-Profiles ermöglicht eine strukturierte Modellierung der Traceability-Beziehungen und unterstützt sowohl Automatisierung als auch langfristige Wartbarkeit der Architektur.

Gleichzeitig ergeben sich gewisse Einschränkungen in Bezug auf Performance und Adaptability. Insbesondere bei der Compliance-Prüfung müssen für jede Policy einzelne Abfragen gegenüber den APIs der Cloud-Provider durchgeführt werden, wodurch eine erhöhte Netzwerklast entstehen kann. Zudem können strukturelle Änderungen an den Traceability-Objekten Anpassungen am Evaluationsalgorithmus erforderlich machen.

Insgesamt weist die Variante eine geringe strukturelle Komplexität bei gleichzeitig hoher Traceability Strength auf. Regulatorische Anforderungen können über Controls und Component-Profiles nachvollziehbar mit konkreten Cloud-Policies verknüpft werden, wodurch eine transparente Abbildung der Compliance-Beziehungen ermöglicht wird.

6.2.1 Bewertung der Quality Attributes

Die Bewertung mittels der Quality-Attribute erzielt einen totalen Score von *3.016* und wird nachfolgend textuell erläutert.

ID	$s(q)$	**Quality-Attribute**
Q1	2	Performance
Q2	3	Portability
Q3	3	Auditability
Q4	3	Automatability
Q5	3	Maintainability
Q6	2	Adaptability
Q7	3	Complexity
Q8	4	Traceability Strength

Tabelle 6.2: Übersicht Bewertung der Auswahl an Quality-Attributes nach Unterunterabschnitt 3.5.2

Performance

Der wichtigste Einflussfaktor auf die Performance ist die Durchführung der Compliance-Prüfung. Für jede Policy, die einem Component-Profile zugeordnet ist, muss eine Anfrage an die API des jeweiligen Cloud-Providers gestellt werden. Da nicht davon ausgegangen werden kann, dass die Provider aggregierte Batch-Abfragen für mehrere Policies unterstützen, entstehen viele einzelne Netzwerkanfragen. Dies führt zu erhöhter Netzwerklast und damit zu einer insgesamt geringeren Performance bei der Evaluation.

Portability

Die Regulations, Controls sowie Component-Profiles sind plattformunabhängig formuliert und enthalten keine Provider-spezifischen Konzepte. Dadurch können dieselben regulatorischen Anforderungen auch auf andere Cloud-Plattformen übertragen werden. Lediglich die Zuordnung von konkreten Policies zu den Component-Profiles muss plattformspezifisch erfolgen. Durch diese Trennung zwischen regulatorischer Ebene und Provider-spezifischem Enforcement wird eine hohe Portability erreicht.

Auditability

Die Traceability-Beziehungen enthalten Metainformationen zu Controls und Component-Profiles, wodurch der Kontext einer Zuordnung nachvollziehbar bleibt. Für eine konkrete Foundation kann somit jederzeit ermittelt werden, welche regulatorischen Anforderungen relevant sind und durch welche Policies diese umgesetzt werden. Mittels der verschiedenen modellierten

Objekte lässt sich auch nachvollziehen, wann und von wem Anpassungen vorgenommen wurden. Dies führt zu einem hohen Grad an Auditability.

Automatability

Die explizite Modellierung der Traceability-Beziehungen ermöglicht eine weitgehende Automatisierung sowohl bei der Erstellung als auch bei der Evaluation der Beziehungen. Policies können automatisiert mit Component-Profiles verknüpft werden und die Compliance-Prüfung kann automatisiert über entsprechende API-Abfragen erfolgen. Auch die initiale Zuordnung von Provider-spezifischen Policies zu Controls kann teilweise automatisiert erfolgen, beispielsweise durch semantische Analyse oder KI-basierte Zuordnungen. Dadurch ergibt sich ein hoher Automatisierungsgrad.

Maintainability

Alle Objekte des Metamodells können versioniert und unabhängig voneinander weiterentwickelt werden. Änderungen an den Component-Profiles, Controls oder Policies erfordern daher in der Regel keine Anpassung der gesamten Modellstruktur. Erweiterungen oder Aktualisierungen können somit mit vergleichsweise geringem Aufwand vorgenommen werden und Änderungen an bestehenden Deployments können durch Migrationen erreicht werden. Dies führt zu einer guten Maintainability.

Adaptability

Strukturelle Änderungen an der Modellierung der Traceability-Beziehungen, wie die Einführung zusätzlicher Beziehungsebenen, können Anpassungen am Evaluationsalgorithmus erforderlich machen. Da der Algorithmus direkt auf der bestehenden Struktur der Beziehungen aufbaut, ist eine Anpassung zwar möglich, jedoch mit zusätzlichem Implementierungsaufwand verbunden. Daher ist die Adaptability nur eingeschränkt gegeben.

Complexity

Die Architektur führt nur jene Objekttypen ein, die für die Modellierung der Traceability-Beziehungen erforderlich sind. Dazu gehören insbesondere Regulations, Controls, Component-Profiles sowie Provider-spezifische Policies. Durch diese begrenzte Anzahl an Konzepten bleibt das Modell übersichtlich und vermeidet unnötige Abstraktionsebenen. Die resultierende Komplexität der Architektur ist daher vergleichsweise gering.

Traceability Strength

Die Beziehungen zwischen Regulations, Controls, Component-Profiles und den entsprechenden Cloud-Policies sind explizit modelliert und eindeutig nachvollziehbar. Dadurch lässt sich für jede regulatorische Anforderung bestimmen, welche Policies zu ihrer Umsetzung beitragen. Ebenso kann für eine konkrete Cloud-Policy nachvollzogen werden, welche regulatorischen Anforderungen sie adressiert. Diese Nachvollziehbarkeit in beide Richtungen führt zu einer hohen Traceability Strength.

6.2.2 Trade-Offs

Explicit Trace Modeling vs. Implicit Derivation

Die Modellierung an den Regulations wie Si001* ist mit den Component-Profiles ziemlich explizit. Die Component-Profiles sind dabei gedanklich an einen möglichen Beschaffungsschritt im 360°-Cockpit angelehnt (siehe Abschnitt 5), in welchem ebenfalls der Bedarf nach Komponenten wie Storage, Network etc. angegeben wird. Durch die Verknüpfung in dieser Variante zwischen Component-Profiles und Controls wird diese bestehende Modellierung bereits integriert.

Governance centric Traceability vs. Engineering centric Traceability

Component-Profiles orientieren sich eher an den Akteuren des Cloud-Experten und des Projektverantwortlichen statt am Compliance-Experten. Betrachtet man den Projektverantwortlichen als Kunden, entspricht dies einer besseren Nutzerzentrierung, auch wenn der Projektverantwortliche selbst bspw. nicht für das Audit verantwortlich ist.

Traceability Depth vs. Model Simplicity

Das Mapping über Component-Profiles erlaubt trotz Fokus auf Components gleichwohl eine Erzeugung von Evidences pro Control. Dies ist möglich, da die Controls im Component-Profile vermerkt sind. Die Component-Profiles erlauben somit zum einen ein Tracing von Foundations und deren Ressourcen zu den Controls und zum anderen, Requirements des Projektverantwortlichen für die Foundation gut zu mappen. Der Requirements-Aspekt ist relevant, da häufig in genau solchen Abstraktionsgruppen wie Datenbanken/Storage, Netzwerk etc., sprich Components, gedacht wird.

Central Traceability Model vs. Federated Traceability Models

Das Modell ist recht geschlossen und erlaubt bei den Component-Profiles keine Definition von eigenen Attributen. Lediglich bei der Deklaration oder Referenzierung von Policies von Cloud-Plattformen ist eine Offenheit vorgesehen. Dies bedeutet, dass bei einer Nutzung dieses Modells von Grund auf bereits vorausgedacht werden muss, um anderen Stakeholder die Nutzung zu ermöglichen. Denkbar sind hier andere Regulations, welche nicht zwangsläufig Component- / Control-basiert denken wie Si001*.

6.3 Evaluation Variante 3

Die Evaluation der Category-zentrierten Variante zeigt eine Architektur, die eine starke strukturelle Vereinfachung durch die Einführung von LZ*-Categories anstrebt. Durch die Bündelung von Policies auf einer höheren Abstraktionsebene kann die Modellkomplexität reduziert und eine plattformübergreifende Anwendbarkeit unterstützt werden. Dies erleichtert insbesondere die Anpassung der Architektur an neue Anforderungen sowie die Erweiterung um zusätzliche Kategorien im Verlauf des Betriebs.

Gleichzeitig führt die Generalisierung der Traceability-Beziehungen dazu, dass Kontextinformationen teilweise verloren gehen. Obwohl weiterhin nachvollzogen werden kann, welche Policies im Rahmen einer Kategorie zur Compliance-Bewertung herangezogen werden, ist die genaue Begründung der Zuordnung einzelner Policies weniger transparent als in Varianten mit stärkerer expliziter Modellierung. Dies wirkt sich insbesondere auf die Auditability sowie die Maintainability der Architektur aus.

Auch hinsichtlich der Performance ergeben sich Einschränkungen, da durch die Kategoriebasierte Bündelung Policies geprüft werden können, die für eine konkrete Foundation oder bestimmte Ressourcentypen nicht unmittelbar relevant sind. Insgesamt positioniert sich die Variante als Engineering-orientierter Ansatz mit hoher Modellvereinfachung und guter Adaptability, die jedoch zugunsten dieser Vereinfachung eine geringere Detailtiefe und Transparenz der Traceability-Beziehungen in Kauf nimmt.

6.3.1 Bewertung der Quality Attributes

Die Bewertung mittels der Quality-Attribute erzielt einen totalen Score von *2.508* und wird nachfolgend textuell erläutert.

ID	$s(q)$	Quality-Attribut
Q1	1	Performance
Q2	3	Portability
Q3	2	Auditability
Q4	2	Automatability
Q5	1	Maintainability
Q6	3	Adaptability
Q7	4	Complexity
Q8	3	Traceability Strength

Tabelle 6.3: Übersicht Bewertung der Auswahl an Quality-Attributen nach Unterunterabschnitt 3.5.2

Performance

Das Mapping der Policies auf eine der LZ*-Categories erlaubt, dass Policies bei der Verifikation gebündelt abgefragt werden können. Wegen der Kategorisierung werden jedoch auch Policies für Ressourcentypen geprüft, die potenziell gar nicht in der Foundation vorkommen. Für jede Policy-Prüfung sind Calls an die APIs der Cloud-Provider nötig. Entsprechend ist die Performance gering.

Portability

Die Abstraktion der LZ*-Category erlaubt, ähnlich wie bei den Component-Profiles, eine Anwendung auf anderen Cloud-Plattformen. Bei einer Nutzung auf einer weiteren Plattform müssen lediglich neue Policies der Category zugeordnet werden.

Auditability

Die LZ*-Category erlaubt es nachzuvollziehen, dass Policies den Categories zugeordnet wurden. Gleichzeitig sinkt jedoch wegen der hohen Anzahl an Controls die Nachvollziehbarkeit, warum bspw. eine gewisse Policy einer Konfiguration einer Category angehört. Die Auditability ist somit tiefer.

Automatability

Die technische Modellierung der LZ*-Category erlaubt eine maschinelle Zuweisung und Anwendung auf die Foundation. Die Zuweisung der Policies zu der Category kann KI-basiert erfolgen, muss jedoch manuell nachgeprüft werden. Die manuelle Prüfung ist hier besonders relevant, da der Kontext, auf dessen Basis beurteilt werden kann, ob eine Policy einer Category angehört, unklarer ist. Dies zeigt sich vor allem im Vergleich mit den Component-Profiles oder Controls, die auf einen genaueren Scope zurückgreifen können.

Maintainability

Die Maintainability ist aufwändiger, da die Zuordnung von Policy zu LZ*-Category evtl. nicht durch das Modell an sich nachvollziehbar ist. Nicht nachvollziehbar meint hierbei, dass bspw. ein Compliance-Officer die Überlegungen, welche zur Zuweisung durch einen anderen Compliance-Officer geführt haben, nicht zwangsläufig verstehen kann.

Adaptability

Die LZ*-Categories können sich an einen neuen oder sich ändernden Bedarf anpassen. So können im Verlauf der Nutzung neue Categories definiert werden, die spezifische Policies fassen. Durch die Abstraktion kann die Migration unabhängig von der Cloud-Plattform erfolgen.

Complexity

Die Modellierung beschränkt sich bewusst lediglich auf die Ebene der LZ*-Category, um eine Vereinfachung anzustreben. Deshalb wird die Reduktion der Complexity hoch bewertet.

Traceability Strength

Trotz der Generalisierung in den verschiedenen Categories kann nachvollzogen werden, was der Compliance-Status einer Control ist. Lediglich der Grund der Zuweisung ist wegen der potenziell grossen Anzahl an Policies in einer Category nicht unbedingt eindeutig.

6.3.2 Trade-Offs

Explicit Trace Modeling vs. Implicit Derivation

Die Variante entspricht einer expliziten Modellierung. Die LZ*-Category dient dabei als Bündelung von Controls zu Policies. Der Gewinn gegenüber einer impliziten Modellierung ergibt sich aus den Abstufungen, welche die Categories ermöglichen. Requirements des Akteurs Projektverantwortlicher können somit direkt das Mass der Anwendung steuern.

Governance centric Traceability vs. Engineering centric Traceability

Die Modellierung orientiert sich stark an dem Akteur des Projektverantwortlichen. Durch die starke Abstraktion der angewendeten Policies und Controls findet eine Vereinfachung statt. Diese Vereinfachung führt dazu, dass der Projektverantwortliche keine detaillierten Requirements mehr machen muss, sondern lediglich eine Aussage in den Kategorien machen kann.

Traceability Depth vs. Model Simplicity

Die Variante nimmt eine ähnliche Abstraktion wie die Component-zentrische Variante vor, abstrahiert jedoch durch die stark reduzierten LZ*-Kategorien noch stärker. Die Details der Traceability-Beziehung werden dadurch nicht direkt beeinflusst. Die Zuordnung zu verschiedenen Ressourcentypen ist jedoch nicht mehr gegeben, da Policies von allen Controls direkt der LZ*-Category zugeordnet werden.

Central Traceability Model vs. Federated Traceability Models

Das Modell erlaubt trotz des zentralen Modells der LZ*-Categories die Definition weiterer Categories, die theoretisch eigenständig zugeteilt und hierarchisch angeordnet werden können. Somit kann sich trotz der zentralen Definition des LZ*-Category-Modells selbst eine gewisse föderative Nutzung ergeben.

6.4 Evaluation Variante 4

Die Evaluation der Provider-nativen Variante zeigt eine Architektur, die bewusst auf zusätzliche Abstraktionsebenen verzichtet und stattdessen die nativen Policy- / Compliance-Mechanismen der jeweiligen Cloud-Plattformen direkt nutzt. Dadurch kann eine sehr hohe Performance bei der Compliance-Auswertung sowie ein hoher Grad an Automatisierbarkeit erreicht werden, da bestehende providerseitige Aggregations- und Evaluationsmechanismen wiederverwendet werden.

Gleichzeitig führt die enge Kopplung an plattformspezifische Konzepte zu Einschränkungen hinsichtlich der Portability und Adaptability der Architektur. Unterschiede in den Policy-Modellen, Metadatenstrukturen und Auswertungsmöglichkeiten zwischen verschiedenen Cloud-Providern erschweren eine konsistente, plattformübergreifende Traceability-Betrachtung. Auch die langfristige Maintainability kann dadurch beeinflusst werden, da Wissen über Provider-spezifische Mechanismen und deren Weiterentwicklung erforderlich ist.

Die strukturelle Komplexität der Architektur bleibt durch den Verzicht auf ein zusätzliches Traceability-Metamodell gering. Gleichzeitig kann jedoch die Nachvollziehbarkeit regulatorischer Zusammenhänge variieren, da die verfügbaren Traceability-Informationen stark von den jeweiligen Plattformfunktionen abhängen. Insgesamt positioniert sich die Variante als stark Engineering-zentrierter Ansatz mit hoher operativer Effizienz, der zugunsten dieser Effizienz eine geringere strategische Unabhängigkeit und Modellkonsistenz in Kauf nimmt.

6.4.1 Bewertung der Quality Attributes

Die Bewertung mittels der Quality-Attribute erzielt einen totalen Score von *2.59* und wird nachfolgend textuell erläutert.

ID	$s(q)$	**Quality-Attribut**
Q1	4	Performance
Q2	1	Portability
Q3	2	Auditability
Q4	4	Automatability
Q5	2	Maintainability
Q6	1	Adaptability
Q7	4	Complexity
Q8	3	Traceability Strength

Tabelle 6.4: Übersicht Bewertung der Auswahl an Quality-Attributen nach Unterunterabschnitt 3.5.2

Performance

Die direkte Nutzung von providerspezifischen Policy-Engines lässt annehmen, dass die Auswertung von bspw. Bundle-Policies bereits optimal erfolgt. Im Gegensatz zu den anderen Varianten, bei denen individuelle Abfragen mit einer gewissen Latenz an die APIs der Cloud-Plattformen vorgenommen werden, entfällt dieser Schritt bei dieser Variante. Die Performance ist somit hoch.

Portability

Die Nutzung von plattformspezifischen Konzepten reduziert die Wahrscheinlichkeit, dass eine Übernahme auf andere Plattformen möglich ist. Trotz ähnlicher Idee bzgl. Enforcement/Policy-Prüfung können die Konzepte zur Umsetzung grundlegend verschieden sein. Die Portability ist somit tief.

Auditability

Cloud-Plattformen wie AWS und Azure bieten out-of-the-Box Services an, mit denen die Zuweisung von bspw. Policies und weitere Aktionen nachvollziehbar sind. Gleichzeitig ist aber nicht garantiert, dass der Grad der Auditability den Anforderungen der Akteure entspricht.

Automatability

IaC*-Tools bieten für bekanntere Cloud-Plattformen wie Azure und AWS häufig Integrationen an. Diese maschinelle Zuweisung von Policies erlaubt später auch das direkte Auslesen des Compliance-Status. Die Automatability ist deshalb hoch.

Maintainability

Die Elemente, welche von den Cloud-Plattformen selbst bereitgestellt werden, besitzen (wie bei den anderen Varianten indirekt) ihren eigenen Lifecycle, was zu einem geringen Aufwand führt.

Die Parität über die Cloud-Plattformen hinweg ohne ein Modell obendrauf aufrechtzuerhalten, ist wahrscheinlich langfristig aufwändiger. Des Weiteren müssen tendenziell mehr Akteure Wissen über die einzelnen Cloud-Plattformen besitzen. Deshalb wird die Maintainability gering bewertet.

Adaptability

Eine fehlende, auf den Cloud-Plattformen aufbauende Abstraktion erlaubt keine Adaptability, wenn dies nicht bereits von der Plattform vorgesehen ist. Basierend auf den verschiedenen Konzepten, welche die Provider verfolgen, wird somit eine geringe Adaptability angenommen.

Complexity

Die Complexity ist wegen der nicht vorhandenen Abstraktion sehr gering. Konsequenzen daraus werden im Aspekt der Maintainability beleuchtet.

Traceability Strength

Mit der Traceability soll das Verknüpfen von Controls zu Ressourcen ermöglicht werden. AWS und Azure ermöglichen es, via Metadaten gewisse Informationen in Policy-Assignments einzubauen, um einen Rückschluss zu ermöglichen. Basierend auf den genannten Cloud-Plattformen wird angenommen, dass auch andere Plattformen Tools bereitstellen, um solche Abfragen zu ermöglichen. Die Annahme ist jedoch, dass eine Aggregierung über Plattformen hinweg nur teilweise möglich ist.

6.4.2 Trade-Offs

Explicit Trace Modeling vs. Implicit Derivation

Die Variante entspricht einer expliziten Modellierung, in der die technischen Mittel der Cloud-Plattformen selbst verwendet werden, um die Regulations abzubilden. Dadurch reduziert sich der Aufwand, ein separates Traceability-Beziehungsmodell zu pflegen. Hingegen ist der Aufwand hoch, die Modellierungen auf den verschiedenen Cloud-Plattformen gleich zu halten.

Governance centric Traceability vs. Engineering centric Traceability

Wegen der Unterschiede von Plattform zu Plattform kann nicht allgemein gesagt werden, wie die Zentrierung aussieht. Für Azure wird durch die Zentralisierung der Policies eher eine Governance-centric Traceability erwartet, welche für den Akteur des Compliance-Officers nutzbarer ist. Bei AWS wird wegen der Verteilung von Policies über Services eher eine Engineering-centric Traceability erwartet.

Traceability Depth vs. Model Simplicity

Die Model Simplicity ist wegen der geringen bzw. nicht existierenden Abstraktion sehr hoch. Gleichzeitig geht wegen der fehlenden Abstraktion die Parität von Informationen über Cloud-Plattformen hinweg verloren. Der Compliance-Officer muss z. B. sicherstellen, dass Auswertungen, welche von einem Cloud-Provider zu einer Control bereitgestellt werden, jenen bei einem anderen Provider ähnlich sind.

Central Traceability Model vs. Federated Traceability Models

Die direkte Nutzung der Policy-Engines der Cloud-Provider macht das Modell, das von den Providern propagiert wird, grundsätzlich zu einem zentralen Modell. Azure erlaubt dem Nutzer, Policy-Initiatives zu erstellen, um Policies kombinieren zu können. Diese haben ein Feld für Metadaten, dort können eigene Informationen beigefügt werden. AWS erlaubt je nach Service mehr oder weniger Spielraum, was das Definieren und Strukturieren von eigenen/built-in Policies anbelangt.

7 Diskussion

Die Diskussion setzt die Ergebnisse der Evaluation der Traceability-Architektur in Perspektive und gibt anschliessend eine Empfehlung für den Use-Case-Kontext des BIT* ab. Mit der Reflexion des Architektur-Ansatzes wird untersucht, ob die Architektur auch ausserhalb dieses Kontexts angewendet werden kann. Die Reflexion der Methodik betrachtet, ob der vorgeschlagene Evaluationsansatz generell anwendbar ist. Die Limitationen der Arbeit sollen die vorhergehenden Kapitel bewusst transparent einordnen.

7.1 Interpretation der Evaluationsergebnisse

Die Scores der verschiedenen Traceability-Architektur-Varianten gemäss Tabelle 7.1 zeigen, dass eine zusätzliche Ebene der Abstraktion nicht zwangsläufig zu einem besseren Ergebnis führt. Dies erstaunt auf den ersten Blick, da eine der Grundideen der Traceability-Architektur darin besteht, einen von den Cloud-Plattformen unabhängigen Implementationsvorschlag zu haben. Gleichwohl schneidet die Provider-native Variante besser ab als die Category-centric Variante. Eine Begründung hierfür könnte sein, dass die Wahl der Abstraktion bewusst getroffen werden muss, um einen effektiven Mehrwert zu generieren. Man denke hier an die sogenannte premature Generalization in der Programmierung, welche in Antizipation anderer Anforderungen Strukturen einführt, die sich womöglich am Ende nicht bewähren. Der Vergleich der Provider-native-Variante mit den anderen Varianten zeigt, dass vor allem der negative Einfluss auf die Performance und die Tatsache, dass die relative Complexity bewertet werden, für einen Provider-nahen Ansatz sprechen. Andererseits führt ein Provider-nativer Ansatz zu einer schlechteren Maintainability in Bezug auf die Aufrechterhaltung einer Vergleichbarkeit zwischen den Cloud-Plattformen. Es zeigt sich somit, dass es relevant ist, die Qualitäts-Attribute in dem Kontext abzuwägen, für den sie relevant sind.

Architektur-Variante	**Score** $s(q)$
Control-centric	3.18
Component-centric	3.016
Category-centric	2.508
Provider-native	2.59

Tabelle 7.1: Bewertung der verschiedenen Architektur-Varianten basierend auf den Qualitäts-Attributen und Trade-Off Kriterien.

Die bewerteten Trade-Off-Dimensionen zeigen ein ähnliches Bild, beispielsweise beim Vergleich der expliziten vs. impliziten Modellierung der Traceability sowie der Traceability Depth vs. Simplicity. Die implizite Ableitung, welche häufig mit einer Model Simplicity der Traceability-Beziehung selbst einhergeht, ist ausreichend, wenn z. B. lediglich eine einfache Aussage über die Compliance gemacht werden soll. Denkbar sind hier die bestehenden Compliance Frameworks und Rulesets, welche bspw. im Azure Defender zur Verfügung stehen. Bei einer Anwendung dieser vordefinierten Sets ist vielleicht weniger das Verständnis der Anwendung als vielmehr die Anwendung an sich relevant, da dies eher von der Cloud-Plattform selbst sichergestellt wird. Auf der anderen Seite ist die Ausprägung der expliziten Trace-Modellierung relevant, wenn bspw. eigene Regulations modelliert werden. Bei dieser Modellierung muss der Compliance-Officer detaillierter nachvollziehen können, wie ein Assessment zustande kommt.

Für die Beurteilung der verschiedenen Varianten wird die Component-centric Variante herangezogen, da sich der PoC* an dieser Variante orientiert. Die Bewertung der Performance zeigt, dass die Prüfung des Compliance-Status einzelner Policies möglich ist, bei einer hohen Anzahl

an Policies jedoch eine spürbare Verzögerung entsteht, da einzelne API Calls erforderlich sind. Basierend auf dieser Erfahrung scheinen die Bewertungen bei den anderen Varianten, die ebenfalls eine Abstraktion verwenden, realistisch. Für die Maintainability hingegen ist der Wert des PoC* mit Vorsicht zu betrachten, da der gesamte Aspekt der Modellierung nicht auf einem production-ready Niveau betrachtet wird. Konkret werden im PoC* Policies nur manuell den Component-Profiles zugeordnet. An dieser Stelle fehlt jedoch die Bewertung darüber, wie dies im Grossbetrieb funktionieren würde. Der vorgeschlagene Ansatz der KI-basierten Zuweisung scheint realistisch, muss jedoch separat betrachtet werden. Diese Art der Zuweisung kann wiederum Implikationen auf die Maintainability haben.

Aussagen über eine konkrete LZ*-Architektur sind basierend auf der Bewertung der Traceability-Architektur nur beschränkt möglich. Ausführungen zur Architektur selbst werden in Unterabschnitt 7.3 gemacht.

7.2 Empfehlung für den BIT Use-Case

Die Konformitätsprüfung der Traceability-Architektur zeigt, dass die Architektur den relevanten Architekturprinzipien des BIT* entspricht. Für den konkreten Use-Case des BIT* wird die Component-centric Variante der Traceability-Architektur empfohlen. Das 360°-Cockpit für den Akteur des Projektverantwortlichen basiert auf der Annahme, dass beim Bestellen des Projekts und der Foundation als Teil der Cloud-Auswahl bereits eine Aussage über die Nutzung der Arten von Cloud-Ressourcen gemacht werden muss. Mit dieser Aussage wird ermittelt, welche Cloud-Plattform für eine Beschaffung den besten Preis offeriert. Im Sinne der Serviceorientierung greift die Component-centric Variante die Bündelung nahe an der Businessdomäne auf. Neben der von Use-Cases getriebenen Begründung kategorisieren auch die Cloud-Provider AWS und Azure ihre Produkte/Angebote in Kategorien. Beispiele sind hier Compute, Container, Identity etc. Die Annahme ist, dass diese Unterteilung auch bei der Berechnung für die Beschaffung berücksichtigt wird, bspw. in den eigenen Cost-Calculators der Cloud-Plattformen. Die Orientierung an dieser groben Unterteilung verspricht des Weiteren, dass der Aufwand für die Zuordnung von Policies zu Component-Profiles geringer ist, da die Cloud-Plattformen eine derartige Zuweisung bereits implizit vorgenommen haben. Die Annahme ist, dass somit mittelfristig der Aufwand für den Betrieb reduziert werden kann. Gleichwohl muss jedoch als Risiko berücksichtigt werden, dass für einen effizienten Betrieb die Nutzung einer semantischen Mapping Engine nahezu notwendig ist. Diese Engine muss regelmässig prüfen, welche Policies auf den Cloud-Plattformen existieren und ob diese basierend auf der Beschreibung für eine Control in einem bestimmten Component-Profile passend sind. Vorstellbar ist hier, einen KI-Agenten zu trainieren, welcher den Grossteil der Arbeit übernimmt. Mittels Human-in-the-Loop, welcher durch den Akteur des Cloud-Experten oder Compliance-Officers wahrgenommen werden kann, wird sichergestellt, dass die Zuordnung zur Control stimmt.

Alternativ zur Component-centric Variante kann auch eine modifizierte Form der Control-centric Variante für den Use-Case verwendet werden. Modifiziert heisst hier, dass ein direktes Mapping von Policies zu den Controls nicht ausreichend ist, da nicht für jede Foundation alle Policies einer Control relevant sind. Als Beispiel darf keine Control in eine nicht-produktive Umgebung übernommen werden, in der der Projektverantwortliche bspw. aus Kostengründen keine Redundanz gewisser Ressourcen möchte. Würde dies in einem produktiven Setup enforced, liesse sich für das nicht-produktive Setup nicht sagen, ob es sich um eine effektive Verletzung der Compliance handelt.

Die Category-centric und Provider-native Variante werden für den Use-Case nicht empfohlen. Die bei der Category-centric Variante angedachte Vereinfachung durch die wenigen Categories sowie die Traceability-Depth versprechen keinen Vorteil für den Akteur des Compliance-Officers

bei der Durchführung eines Audits. Die Provider-native Variante hat einen ähnlichen initialen Aufwand wie die vorher genannten Varianten, da es sich bei Si001* höchstwahrscheinlich um eine den Providern unbekannte Regulation handelt. Mittelfristig wird vermutet, dass die Maintenance höher ist, da die gleiche Anwendung über die Cloud-Plattformen hinweg manuell sichergestellt wird und nicht über eine der Abstraktionen, wie sie in den anderen Varianten beschrieben wird, gelöst werden kann. Würde statt Si001* hingegen eine weitverbreitete Regulation wie ISO27001 verwendet, würde die Nutzung des Built-in-Mappings empfohlen, wie es die meisten Cloud-Plattformen bereits anbieten. Die Maintenance würde hierbei vollständig an die Cloud-Plattform entfallen.

7.3 Reflexion des Architektur-Ansatzes

Die Traceability-Architektur (siehe Abbildung 4.3) baut auf dem Konzept der Regulations und Controls auf. Dieses findet sich nicht nur bei Si001*, sondern auch bei Regulations von Organisationen wie dem NIST*. Ebenso ist das Konzept der Policy, wie es bei Azure und AWS bekannt ist, auch auf anderen Cloud-Plattformen bekannt, wenn auch in unterschiedlicher Form. Somit sind alle Aspekte der Traceability-Architecture und die Konstruktion einer Traceability-Beziehung auch in einem Kontext ausserhalb des Si001* möglich.

Die LZ*-Architektur selbst stützt sich auf die Capabilities (siehe Abbildung 3.3), welche zur Identifizierung der relevanten LZ*-Components verwendet werden. Die Traceability-Architektur konnte als Building-Block aus der Capability-Map zusammen mit dem BIT* abgeleitet werden. Die Validierung der Traceability-Architektur zeigt somit indirekt, dass eine sukzessive Ableitung von Capabilities hin zu Teilen einer LZ*-Architektur möglich ist. Mit dem LZ*-Blueprint, welcher im PoC* verwendet wird, zeigt sich, wie eine Implementation weiterer Capabilities aussehen könnte. Ein Prozess, um eine umfassendere LZ*-Architektur aufzubauen, könnte wie folgt aussehen:

1. Abhängigkeiten zwischen Level 2 Capabilities identifizieren
2. Building-Blocks für Auswahl an Capabilities
3. Architektur definieren welche die Capability implementiert

Basierend auf dem Outcome der verschiedenen Architekturen, welche einen Einfluss auf die LZ*-Architektur haben, können dann Blueprints für LZ* auf Ebene der Lösungsarchitektur erstellt werden.

7.4 Reflexion der Evaluationsmethodik

Allgemein kann gesagt werden, dass sich die grobe Orientierung der Evaluation an bestehenden EA* Frameworks wie TOGAF* als sinnvoll erweist, um eine Struktur für die Entwicklung von Architekturen zu erhalten. Gleichzeitig handelt es sich aber nicht um ein Mittel, um objektive Architekturentscheidungen treffen zu können. Die Verwendung von Architekturprinzipien zur Konformitätsprüfung in Unterunterabschnitt 3.5.1 zeigt dies bereits implizit. In dem Prozess steht nicht im Fokus, ob ein Architekturprinzip plakativ gesprochen gut oder schlecht ist, sondern ob es für den Kontext passend ist. Für diese Arbeit werden dafür die Architekturprinzipien des BIT* sowie externe Architekturprinzipien aus der Literatur verwendet. Die Auswahl der externen Prinzipien erfolgt dabei im Multi-Cloud-Kontext. Bei der Prüfung der Konformität der Traceability-Architektur zeigt sich, dass die kombinierte Anwendung von Architekturprinzipien die Qualität der Architektur allein dadurch erhöht, dass mehr Betrachtungswinkel einbezogen werden. Dies geschieht z. B. durch die Argumentation, warum die Traceability-Architektur ein gewisses Prinzip erfüllt oder nicht erfüllt. Gleichzeitig muss jedoch berücksichtigt werden, dass

der Ausschluss von Prinzipien ebenso gut begründet werden muss. Die Begründung für die Auswahl der Architekturprinzipien in dieser Arbeit ist dabei verbesserungsfähig, da sie subjektiv aus der Erfahrung der Autoren erfolgt und nicht empirisch belegt wird. Für eine bessere Verankerung wären hier z. B. Fallstudien denkbar. Trotz der zu Beginn erwähnten Recherche nach allgemeinen Architekturprinzipien in der Literatur für den Multi-Cloud-Kontext wäre demnach ebenfalls eine Recherche für die Auswahl sinnvoll.

Der Vergleich der Varianten anhand von Qualitäts-Attributen zusammen mit den Trade-Off-Kriterien erweist sich für die Traceability-Architektur als sinnvoll. Quality-Attributes selbst sind ein bekanntes Mittel zur Bewertung von Konzepten. In der Arbeit wird, ähnlich wie bei den Architekturprinzipien, auch für die Qualitäts-Attribute eine Auswahl vorgenommen. Es zeigt sich, dass die Auswahl der Attribute nach dem Anwendungskontext sinnvoll ist, da durch die Beschreibung der Attribute gewissermassen die Eigenschaften der Architektur-Varianten aufgezeigt werden können. Da für einen Anwendungskontext nicht alle Eigenschaften gleichermassen relevant sind, ist eine Auswahl gerechtfertigt. Für einen Vergleich über Varianten hinweg verwendet die Methodik einen Score. Der Score selbst hat sich als praktisch erwiesen, wirft jedoch die Frage auf, wie die Relevanz der Attribute hinreichend in der Bewertung abgebildet wurde. In diesem Fall wurden die Gewichte der Attribute basierend auf dem Anwendungskontext bestimmt, um ihre Relevanz zu betonen. Es wird jedoch zu wenig beleuchtet, welcher Score pro Attribut wie bei einer Cost-Benefit-Analyse «gut genug» ist. Für die aktuelle Evaluation ist die Annahme, dass der maximale Score eines Attributes automatisch der beste Wert ist. Dadurch kann der totale Score pro Variante ein falsches Bild zeichnen. Um diesen Effekt zu reduzieren, werden ebenfalls Trade-Off-Kriterien verwendet. Die Trade-Off-Kriterien ermöglichen wie angedacht, die Konfliktfelder von Eigenschaften der Traceability-Architektur-Varianten besser zu beleuchten, und erweisen sich somit als sinnvoll. Bei der Auswahl der Qualitäts-Attribute und Trade-Off-Kriterien zeigt sich jedoch, dass diese abgestimmt erfolgen muss, um eine Überlappung zu vermeiden. Mit Überlappung ist hier nicht eine Überlappung innerhalb der Trade-Off-Kriterien oder Quality-Attributes gemeint, sondern, dass die beiden Seiten eines Trade-Off-Kriteriums nicht bereits als Quality-Attributes selbst vorkommen sollen. Bei einer solchen Überlappung wird die Funktion der Trade-Offs geschwächt, Konfliktfelder zu betonen, welche über die Quality-Attributes hinausgehen.

Fehlend in der Evaluation hingegen ist eine detaillierte Methodik zum Scoping/der Auswahl von Architekturteilen für eine LZ*. Die Wahl der Traceability-Architektur als Beispiel für weitere Architekturen erfolgt aufbauend auf der Capability-Map, welche mit Use-Cases für den Anwendungskontext greifbarer gemacht werden soll. Bei diesem Ansatz stellt sich die Frage, ob der Einbezug des Anwendungskontexts notwendig ist. Denkbar wäre hier, dass analog zur Traceability-Architektur weitere Architekturen mit anderen Use-Cases aufgebaut werden könnten. Dies würde bedeuten, dass sich die Organisation, welche eine LZ*-Architektur konzeptionieren möchte, zuerst über ihre Use-Cases bewusst werden muss, bevor eine Konzeption möglich ist. Als Ausgangspunkt könnte hierbei die Capability-Map in Abbildung 3.3 verwendet werden, wie es in dieser Arbeit für die Traceability-Architektur gemacht wurde.

Trotz des erkannten Verbesserungspotentials für die Prüfung der Architektur und die Evaluation der Varianten zeigt sich, dass die Verwendung einer Capability-Map als Darstellung der Absicht des Businesses ein wertvoller Ansatz ist. Mit diesem kann der Aufbau einer LZ*-Architektur systematischer angegangen werden, indem technische Konzeptionen/Implementationen durch den Business-Need gerechtfertigt werden können. Dadurch kann zum einen geprüft werden, ob alle für eine LZ* relevanten Aspekte berücksichtigt wurden, und zum anderen können Components identifiziert werden, welche aus Sicht des Businesses nicht relevant sind.

7.5 Limitationen der Arbeit

Mit dem Fokus auf die Traceability-Architecture wird nur eine vereinzelte Implementation/Architektur für gewisse Capabilities gezeigt. Die Struktur der LZ*-Architektur und die Idee der Ableitung basierend auf der Capability-Map funktionieren zwar mit dem Beispiel der Traceability-Architektur, wodurch sich aber keine Allgemeingültigkeit ableiten lässt. Dafür müssten zusätzliche Capabilities in Form weiterer Architekturen, Building-Blocks etc. erstellt werden. Hierzu kann bspw. eine angepasste Form der Evaluationsmethodik verwendet werden. Können neben der Traceability-Architektur weitere Elemente für eine LZ*-Architektur abgeleitet werden, liesse sich dieser Ansatz festigen. Dies bedeutet, dass diese Arbeit keine LZ*-Architektur auf der Ebene einer Lösungsarchitektur, sondern lediglich auf EA*-Ebene vorschlägt.

Ein weiterer Punkt ist der sachgemäss kleine Scope des PoC* in Abschnitt 5. Der PoC* zeigt neben der technischen Implementation einer Traceability-Architektur vor allem auf, wie eine Implementation plattformübergreifend aussehen könnte. Dies wird exemplarisch durch die partielle Implementation für die Cloud-Plattformen Azure und AWS gezeigt. Auch wenn hierbei mit Unterabschnitt 3.2 die Eigenschaften anderer Cloud-Plattformen berücksichtigt wurden, müsste für eine genauere Aussage bspw. der PoC* für weitere Plattformen implementiert werden. Dadurch liessen sich die Aspekte der Adaptability oder Portability besser bestimmen.

Die Bewertung der Traceability-Architektur stützt sich trotz des PoC* vor allem auf theoretische Annahmen. Dies betrifft vor allem Aspekte wie die Maintainability, welche nicht nur von der Architektur-Variante selbst, sondern auch stark vom Anwendungskontext abhängen. Das bedeutet, dass sich auf einem höheren Abstraktionslevel wie in der Arbeit zwar Aussagen machen lassen, diese jedoch empirisch belegt werden müssten. Die Limitation hier ist somit eine ähnliche wie jene für die LZ*-Architektur selbst.

8 Abschluss

Das Ziel dieser Arbeit ist es, die relevanten Komponenten sowie eine beispielhafte Struktur einer LZ* aufzuzeigen. Die Ableitung einer groben LZ*-Architektur zeigt, dass eine Orientierung am Business-Need ein wichtiger Faktor ist, um eine nutzbare Architektur zu erhalten. Dafür wird in dieser Arbeit innerhalb des Konzepts einer EA* eine Capability-Map erstellt. Mit einem Teil der Capabilities wird anschliessend eine Traceability-Architektur als mögliche Ausprägung einer LZ*-Architektur formuliert und evaluiert. Die Evaluation zeigt, dass sich basierend auf Kriterien wie Architekturprinzipien, Qualitäts-Attributen sowie Trade-Off-Dimensionen die component-centric Variante der Traceability-Architektur für den Use-Case des BIT* eignet.

8.1 Beantwortung der Forschungsfragen

Als Zusammenfassung der Evaluation und Diskussion werden die relevanten Inhalte für die anfänglich definierten Forschungsfragen formuliert.

Welche Komponenten machen eine Landingzone für eine Hybrid-Multi-Cloud-Umgebung aus?

Die LZ* wird in dieser Arbeit als technisches Gerüst betrachtet, welches die verschiedenen Bedürfnisse des Businesses hinsichtlich einer einheitlichen Governance abbildet. Die Komponenten und deren Scope in der LZ* werden somit durch das Business definiert. Diese Arbeit verwendet dafür eine Capability-Map (siehe Abbildung 4.1 (Reduzierte technische Capability Map mit Level 1 Capabilities. Vollständige Map in Abbildung 3.3, eigene Abbildung)) und definiert folgende Komponenten: «Identity and Access Governance», «Security Governance», «Policy and Compliance Lifecycle Governance», «Network and Connectivity», «Data and Secret Governance», «Observability and Operational Insight Governance» sowie «Cost Governance». Basierend auf diesen Komponenten lässt sich eine LZ*-Architektur und somit auch eine LZ* modellieren.

Wie sieht eine geeignete technische Architektur einer Landingzone für eine Hybrid-Multi-Cloud-Umgebung aus und was muss sie erfüllen?

Eine LZ*-Architektur auf EA*-Ebene muss für die Multi-Cloud-Umgebung gewisse Architekturprinzipien erfüllen. Diese fokussieren sich vom Charakter her auf Plattformunabhängigkeit und vor allem auf Konformität mit der EA* der Organisation, für welche die LZ*-Architektur entworfen wird. In dieser Arbeit werden die Prinzipien für den Use-Case des BIT* in die folgenden Kategorien unterteilt: *Governance* (mit Compliance with Law and Policies, Data Governance), *Abstraction and Plattform Independency* (mit Cloud Native by Default, Architectural Abstraction Principal, Product Neutrality), *Structural Architecture* (mit Serviceorientation, Modular Decomposition Principle, Reference Model Principle, Standardisation Automation) und *Security-Architecture* (Security by Design, Defense-in-Depth, Security Reconfigurability Principle). Basierend auf diesen Prinzipien wird die Grundarchitektur für eine LZ* definiert (siehe Modellierung mit ArchiMate in Abbildung 4.2). In dieser Architektur wird der LZ*-Service als Technology Service modelliert, welcher thematisch eine der Capabilities zusammenfasst. Die Ausprägungen hiervon sind die LZ*-Components, wie in der ersten Forschungsfrage beschrieben. LZ*-Components können dabei ebenfalls als shared Component vorkommen. Die Implementation auf den Plattformen wird durch die jeweiligen Plattform-Components dargestellt.

8.2 Beitrag der Arbeit

Diese Arbeit schlägt eine LZ*-Architektur auf EA*-Ebene vor und zeigt mit einer Traceability-Architecture auf, wie eine Teil-Implementation einer LZ* aussehen könnte. Für die allgemeine Ausarbeitung einer LZ* propagiert diese Arbeit einen Business-driven Ansatz, bei dem mittels einer Capability-Map einzelne LZ*-Komponenten abgeleitet werden.

8.3 Ausblick

Diese Arbeit setzt sich mit der Architektur einer LZ* auf der Ebene der EA* auseinander. Für eine direkte Anwendung ist jedoch eine detaillierte Ausarbeitung in Form von Lösungsarchitekturen und entsprechenden Implementationen nötig. Analog zur Traceability-Architektur könnten mit den verbleibenden Capabilities neue Artefakte zur Implementierung abgeleitet werden. Insbesondere durch die Anwendung in einer Unternehmensumgebung liessen sich Erkenntnisse sammeln, wie und ob der vorgeschlagene Ansatz in der Realität funktioniert. Des Weiteren kann die Auswahl geeigneter Architekturprinzipien für die LZ* durch einen eher empirischen Ansatz gehärtet werden.

9 Hilfsmittel

Hilfsmittel	Beschreibung
Elicit https://elicit.com/	Elicit ist ein AI basiertes Tool für die Literaturecherche
Scite https://scite.ai/	Scite ist ein AI basiertes Tool für die Literaturecherche
Swisscovery https://swisscovery.slsp.ch/	Swisscovery ist eine allgemeine Suchmaschine für digitale wie physische Medien unter anderem im Raum Schweiz
ChatGPT https://chatgpt.com/	ChatGPT ist ein multipurpose LLM
Base44 https://base44.com/	Base44 ist eine AI-powered, no-code Plattform um Web-Applikationen zu erstellen
Github Copilot https://github.com/copilot	GitHub Copilot ist ein KI-Coding-Assistent, der Entwicklern hilft, Code schneller und mit weniger Aufwand zu schreiben.

Tabelle 9.1: Verwendete Hilfsmittel

Die genannten Hilfsmittel werden an folgenden Stellen im Dokument eingesetzt. Die Nutzung des Begriffs *Challengen* bedeutet, dass das Tool als Reflexions- und Diskussionshilfe zur kritischen Einordnung eigener Ideen im Sinne eines Sparringpartners verwendet wird.

Hilfsmittel	Verwendung	Betroffene Stelle
Elicit	Recherchieren von Literatur zu LZ*, Komponenten etc.	Unterabschnitt 3.2, ganzes Kapitel Appendix C, erzeugte Literaturrecherche
Scite	Recherchieren von Literatur zu LZ*, Komponenten etc.	Unterabschnitt 3.2, ganzes Kapitel
Swisscovery	Recherchieren von Literatur zu LZ*, Komponenten etc.	Unterabschnitt 3.2, ganzes Kapitel
ChatGPT	Challengen beim Erstellen der Kapitelstruktur	Ganzes Dokument
Base44	Generieren von UI-Mockups für den PoC* zur Veranschaulichung eins möglichen Governance-Cockpits	Screenshots von generiertem Mockup in Abbildung 3.7, Abbildung 3.8. Prompt und weitere Abbildungen in Appendix A
ChatGPT	Sprachliche Überarbeitung und Reformulierung von eigenen Textentwürfen	Abschnitt 1, ganzes Kapitel
ChatGPT	Generieren basierend auf restlichem selbst erstellten Inhalten	Abstract, ganzer Abschnitt
ChatGPT	Challengen bei der Platzierung vom Vorgehen im ADM* Cycle	Tabelle 3.2
ChatGPT	Challengen bei der Umwandlung von NFR* in Capabilities	Unterabschnitt 3.4 in der Capability-Map in Abbildung 3.3, Abbildung 3.4

Hilfsmittel	**Verwendung**	**Betroffene Stelle**
ChatGPT	Ausformulieren der Hauptabläufe	item 3.5, item 3.6
Elicit	Recherchieren von Architektur-Prinzipien	Tabelle 3.8 Appendix D, erzeugte Literaturrecherche Appendix E, erzeugte Literaturrecherche
Elicit	Recherchieren von Quality-Attributen	Tabelle 3.9 Appendix F, erzeugte Literaturrecherche
Elicit	Recherchieren von Trade-Off Dimensionen	Tabelle 3.12 Appendix G, erzeugte Literaturrecherche
ChatGPT	Challengen des Evaluationsvorgehen für Konformitätsprozess und Varianten der Traceability-Architekturen, sowie mathematischer Formulierungen	Unterunterabschnitt 3.5.1, ganzes Kapitel Unterunterabschnitt 3.5.2, ganzes Kapitel Tabelle 3.13, Tabelle 3.14
ChatGPT	Challengen von Kriterien zur Auswahl des PoC*	Unterabschnitt 3.6, ganzes Kapitel
ChatGPT	Generieren von Prompt für Base44 Mock-Up Generierung	Appendix A
ChatGPT	Challengen der Traceability-Architektur Varianten und Anwendung von Archi-Mate Notation	Abschnitt 4, ganzes Kapitel Abbildung 4.2, Abbildung 4.3, Abbildung 4.6, Abbildung 4.9, Abbildung 4.12, Abbildung 4.15
GitHub Copilot	Unterstützen beim Implementieren des PoC* in VSCode IDE mit Copilot Extension. Indirekt durch die gezeigten Code-Fragmente	Abschnitt 5, Code Fragment im ganzen Kapitel
ChatGPT	Generieren der Zusammenfassung basierend auf selbst geschriebener Evaluation der Trade-Off Kriterien und Qualitäts-Attribute der entsprechenden Varianten	Anfang Unterabschnitt 6.1 bis vor Beginn Unterunterabschnitt 6.1.1 Anfang Unterabschnitt 6.2 bis vor Beginn Unterunterabschnitt 6.2.1 Anfang Unterabschnitt 6.3 bis vor Beginn Unterunterabschnitt 6.3.1 Anfang Unterabschnitt 6.4 bis vor Beginn Unterunterabschnitt 6.4.1
ChatGPT	Berechnen der Scores der Traceabilityarchitektur-Varianten mittels generierten Script in Appendix B	Unterabschnitt 6.1, Unterabschnitt 6.2, Unterabschnitt 6.3, Unterabschnitt 6.4
ChatGPT	Challengen der Strukturierung bzw. logischen Aufbau	Unterabschnitt 8.1, ganzes Kapitel Abschnitt 7, ganzes Kapitel
GitHub Copilot	Challengen der Formulierung und Wortwahl	Abschnitt 4, ganzes Kapitel
GitHub Copilot	Prüfung der Grammatik und Syntax	Ganzes Dokument

Hilfsmittel	Verwendung	Betroffene Stelle

Tabelle 9.2: Nutzung der Hilfsmittel in der Arbeit

Quellenverzeichnis

AG, S. (2024, 1. Januar). *Enterprise-grade hybrid and multi-cloud strategies*. Verfügbar 29. Dezember 2025 unter https://learning.oreilly.com/library/view/enterprise-grade-hybrid-and/9781804615119/

Alibaba. (2026). *Landing Zone - Alibaba Cloud*. Verfügbar 11. März 2026 unter https://www.alibabacloud.com/de/solutions/landing-zone

Ardagna, D., Di Nitto, E., Mohagheghi, P., Mosser, S., Ballagny, C., D'Andria, F., Casale, G., Matthews, P., Nechifor, C.-S., Petcu, D., Gericke, A., & Sheridan, C. (2012). MODAClouds: A model-driven approach for the design and execution of applications on multiple Clouds. *2012 4th International Workshop on Modeling in Software Engineering (MISE)*, 50–56. https://doi.org/10.1109/mise.2012.6226014

Arul, K. (2022). Data Engineering Challenges in Multi-cloud Environments: Strategies for Efficient Big Data Integration and Analytics. *International Journal of Scientific Research and Management (IJSRM)*, *10*(6). https://doi.org/10.18535/ijsrm/v10i6.ec08

AWS. (2026a). *Was ist AWS Control Tower?* Verfügbar 12. März 2026 unter https://docs.aws.amazon.com/de_de/controltower/latest/userguide/what-is-control-tower.html

AWS. (2026b). *Was ist eine landing zone?* Verfügbar 11. März 2026 unter https://docs.aws.amazon.com/de_de/prescriptive-guidance/latest/migration-aws-environment/understanding-landing-zones.html

Banijamali, A., Heisig, P., Kristan, J., Kuvaja, P., & Oivo, M. (2019). Software Architecture Design of Cloud Platforms in Automotive Domain: An Online Survey. *2019 IEEE 12th Conference on Service-Oriented Computing and Applications (SOCA)*, 168–175. https://doi.org/10.1109/soca.2019.00032

Berlato, S., Carbone, R., Lee, A. J., & Ranise, S. (2020). Exploring Architectures for Cryptographic Access Control Enforcement in the Cloud for Fun and Optimization. *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security*, 208–221. https://doi.org/10.1145/3320269.3384767

Bhargav Sai Pillati. (2025). Leading through Complexity: Decision-Making Frameworks for Multi-Cloud Integration and Platform Scalability. *Journal of Information Systems Engineering and Management*, *10*(58), 1158–1164. https://doi.org/10.52783/jisem.v10i58s.12801

BK, B. (2025, 25. Oktober). *Public Clouds Bund*. https://www.bk.admin.ch/bk/de/home/digitale-transformation-ikt-lenkung/bundesarchitektur/cloud/public-clouds-bund.html

Casola, V., De Benedictis, A., Rak, M., & Villano, U. (2018). Security-by-design in multi-cloud applications: An optimization approach. *Information Sciences*, *454-455*, 344–362. https://doi.org/10.1016/j.ins.2018.04.081

Elicit. (2025, 8. November). *What are the key architectural components and design principles for estab- lishing a secure and efficient cloud landing zone?* https://elicit.com

Elicit's source for papers. (2025, 30. Dezember). Verfügbar 30. Dezember 2025 unter https://support.elicit.com/en/articles/553025

Emmanni, P. S. (2020). Architecting Cloud-Native Applications for Multi-Cloud Environments. *International Journal of Science and Research (IJSR)*. https://doi.org/10.21275/sr24401230608

Erl, T., Puttini, R., & Mahmood, Z. (2013). Cloud Computing: Concepts, Technology & Architecture.

Gartner. (2013, 23. Januar). *Gartner Identifies Five Ways to Migrate Applications to the Cloud*. Verfügbar 10. November 2025 unter https://web.archive.org/web/20130123224850/http://www.gartner.com/newsroom/id/1684114

Ghazaryan, E. (2025). Comparative Analysis of Cloud Deployment Models: Public, Private, Hybrid, and Multi-Cloud. *International Journal of Engineering and Computer Science*, *14*(7), 27572–27578. https://doi.org/10.18535/ijecs.v14i07.5193

Gibilisco, G. (2016). A methodology and a tool for QoS-oriented design of multi-cloud applications.

Google. (2026, 2. Januar). *Design der Landing-Zone in Google Cloud*. Verfügbar 11. März 2026 unter https://docs.cloud.google.com/architecture/landing-zones?hl=de

Gurram, S. (2025). Multi-Cloud Architectures: Principles, Implementation and Strategic Benefits. *International Journal of Advanced Research in Science, Communication and Technology*. https://doi.org/10.48175/ijarsct-25526

Hajjat, M., Sun, X., Sung, Y.-W. E., Maltz, D., Rao, S., Sripanidkulchai, K., & Tawarmalani, M. (2010). Cloudward bound. *Proceedings of the ACM SIGCOMM 2010 conference*, 243–254. https://doi.org/10.1145/1851182.1851212

Hoppe, F., & Dätwyler, B. (2025, 1. März). *Cloud Compliance in einer multi-provider* (Projektbericht ip5 Nr. 25FS_IMVS29).

IBM. (2026, 20. Januar). *Overview of landing zone deployable architectures*. Verfügbar 11. März 2026 unter https://cloud.ibm.com/docs/secure-infrastructure-vpc?topic=secure-infrastructure-vpc-overview

Joukov, N., & Shorokhov, V. (2013). Cloud Interoperability via Quick Enterprise Applications Re-Builds. *Proceedings of the 3rd International Conference on Cloud Computing and Services Science*, 575–580. https://doi.org/10.5220/0004502205750580

Kratzke, N., & Peinl, R. (2016). ClouNS - a Cloud-Native Application Reference Model for Enterprise Architects. *2016 IEEE 20th International Enterprise Distributed Object Computing Workshop (EDOCW)*. https://doi.org/10.1109/EDOCW.2016.7584353

Laheri, R. (2025). Designing secure and scalable cloud infrastructures using azure landing zones. *Journal of Information Systems Engineering and Management*, *10*(49), 1116–1126. https://doi.org/10.52783/jisem.v10i49s.10052

McCabe, T. (2024, 1. September). *Fundamentals of enterprise architecture*. Verfügbar 29. Dezember 2025 unter https://learning.oreilly.com/library/view/fundamentals-of-enterprise/9781098159368/

Menzel, M., & Ranjan, R. (2011). CloudGenius: Automated Decision Support for Migrating Multi-Component Enterprise Applications to Clouds. *arXiv.org*.

Microsoft. (2025, 15. Dezember). *What is an Azure landing zone?* Verfügbar 11. März 2026 unter https://learn.microsoft.com/de-ch/azure/cloud-adoption-framework/ready/landing-zone/

Mirsalari, S. R., & Ranjbarfard, M. (2020). A model for evaluation of enterprise architecture quality. *Evaluation and Program Planning*, *83*, 101853. https://doi.org/10.1016/j.evalprogplan.2020.101853

Mohammad Asad Hussain. (2025). A comparative analysis of cloud migration strategies for enterprise systems architecture. *World Journal of Advanced Engineering Technology and Sciences*, *15*(2), 747–756. https://doi.org/10.30574/wjaets.2025.15.2.0622

Mulder, J. (2020a, 1. Dezember). *Multi-cloud architecture and governance*. Verfügbar 9. November 2025 unter https://learning.oreilly.com/library/view/multi-cloud-architecture-and/9781800203198/

Mulder, J. (2020b, 1. Dezember). *Multi-cloud architecture and governance*. Verfügbar 29. Dezember 2025 unter https://learning.oreilly.com/library/view/multi-cloud-architecture-and/9781800203198/

Orban, S. (2016, 1. November). *6 Strategies for Migrating Applications to the Cloud | AWS Cloud Enterprise Strategy Blog* [Section: Adoption]. Verfügbar 10. November 2025 unter https:

//aws.amazon.com/blogs/enterprise-strategy/6-strategies-for-migrating-applications-to-the-cloud/

Orban, S. (2017, 26. April). *Four Key Practices for Profiling Your Enterprise Application Portfolio and Accelerating Your Migration to Cloud | AWS Cloud Enterprise Strategy Blog* [Section: Enterprise Strategy]. Verfügbar 10. November 2025 unter https://aws.amazon.com/blogs/enterprise-strategy/four-key-practices-for-profiling-your-enterprise-application-portfolio-and-accelerating-your-migration-to-cloud/

Papaioannou, A., & Magoutis, K. (2013a). An Architecture for Evaluating Distributed Application Deployments in Multi-clouds. *2013 IEEE 5th International Conference on Cloud Computing Technology and Science*, 547–554. https://doi.org/10.1109/cloudcom.2013.79

Papaioannou, A., & Magoutis, K. (2013b). An Architecture for Evaluating Distributed Application Deployments in Multi-clouds. *2013 IEEE 5th International Conference on Cloud Computing Technology and Science*, 547–554. https://doi.org/10.1109/cloudcom.2013.79

Papazoglou, M. (2012). Cloud Blueprints for Integrating and Managing Cloud Federations. *Software Service and Application Engineering*. https://doi.org/10.1007/978-3-642-30835-2_8

Paradigm, V. (2025, 30. Dezember). *TOGAF ADM Tutorial* [TOGAF ADM Tutorial]. Verfügbar 30. Dezember 2025 unter https://www.visual-paradigm.com/guide/togaf/togaf-adm-tutorial/

Quint, P.-C., & Kratzke, N. (2018). Towards a Lightweight Multi-Cloud DSL for Elastic and Transferable Cloud-native Applications. *Proceedings of the 8th International Conference on Cloud Computing and Services Science*, 400–408. https://doi.org/10.5220/0006683804000408

Sai, B., & Pillati. (2025). Leading through Complexity: Decision-Making Frameworks for Multi-Cloud Integration and Platform Scalability. *Journal of Information Systems Engineering & Management*. https://doi.org/10.52783/jisem.v10i58s.12801

Saraswat, M., & Tripathi, R. (2020). Cloud Computing: Comparison and Analysis of Cloud Service Providers-AWs, Microsoft and Google [Conference Name: 2020 9th International Conference System Modeling and Advancement in Research Trends (SMART) ISBN: 9781728189086 Place: Moradabad, India]. *2020 9th International Conference System Modeling and Advancement in Research Trends (SMART)*, 281–285. https://doi.org/10.1109/SMART50582.2020.9337100

Security, A. C. (n. d.). *Modell der geteilten Verantwortung – Amazon Web Services (AWS)* [Amazon Web Services, Inc.]. Verfügbar 18. März 2026 unter https://aws.amazon.com/de/compliance/shared-responsibility-model/

Srikanth Gurram. (2025). Multi-cloud architectures: Principles, implementation and strategic benefits. *International Journal of Advanced Research in Science, Communication and Technology*, 151–158. https://doi.org/10.48175/IJARSCT-25526

Tritsiniotis, E. D. (2016). Get ready for the Cloud: Tailoring Enterprise Architecture for Cloud ecosystems.

Vorgabenportal BIT – Homepage. (2025, 30. Dezember). Verfügbar 30. Dezember 2025 unter https://portal.collab.admin.ch/sites/vorgabenportal/SitePages/Home.aspx

vpadmin. (2025, 21. Januar). *Comprehensive guide to business capability planning in TOGAF* [Visual paradigm TOGAF]. Verfügbar 16. März 2026 unter https://togaf.visual-paradigm.com/2025/01/21/comprehensive-guide-to-business-capability-planning-in-togaf/

Zhang, L.-J., & Zhou, Q. (2009). CCOA: Cloud Computing Open Architecture. *2009 IEEE International Conference on Web Services*. https://doi.org/10.1109/icws.2009.144

Zhao, H., Benomar, Z., Pfandzelter, T., & Georgantas, N. (2022). Supporting Multi-Cloud in Serverless Computing. *2022 IEEE/ACM 15th International Conference on Utility and Cloud Computing (UCC)*. https://doi.org/10.1109/ucc56403.2022.00051

Eigenständigkeitserklärung

Ich (wir) erkläre(n) hiermit, dass ich (wir) den vorliegenden Leistungsnachweis selber und selbständig verfasst habe(n),

- dass ich (wir) sämtliche nicht von mir (uns) selber stammenden Textstellen und anderen Quellen wie Bilder etc. gemäss gängigen wissenschaftlichen Zitierregeln[3] korrekt zitiert und die verwendeten Quellen klar sichtbar ausgewiesen habe(n);
- dass ich (wir) in einer Fussnote oder einem Hilfsmittelverzeichnis alle verwendeten Hilfsmittel (KI-Assistenzsysteme wie Chatbots[4], Übersetzungs-[5] Paraphrasier-[6] oder Programmierapplikationen[7]) deklariert und ihre Verwendung bei den entsprechenden Textstellen angegeben habe(n);
- dass ich (wir) sämtliche immateriellen Rechte an von mir (uns) allfällig verwendeten Materialien wie Bilder oder Grafiken erworben habe(n) oder dass diese Materialien von mir (uns) selbst erstellt wurde(n);
- dass das Thema, die Arbeit oder Teile davon nicht bei einem Leistungsnachweis eines anderen Moduls verwendet wurden, sofern dies nicht ausdrücklich mit der Dozentin oder dem Dozenten im Voraus vereinbart wurde und in der Arbeit ausgewiesen wird;
- dass ich mir (wir uns) bewusst bin (sind), dass meine (unsere) Arbeit auf Plagiate und auf Drittautorschaft menschlichen oder technischen Ursprungs (Künstliche Intelligenz) überprüft werden kann;
- dass ich mir (wir uns) bewusst bin (sind), dass die Hochschule für Technik FHNW einen Verstoss gegen diese Eigenständigkeitserklärung bzw. die ihr zugrundeliegenden Studierendenpflichten der Studien- und Prüfungsordnung der Hochschule für Technik verfolgt und dass daraus disziplinarische Folgen (Verweis oder Ausschluss aus dem Studiengang) resultieren können.

Windisch, 20. März 2026

Name: Frithjof Hoppe

Unterschrift: Hoppe Frithjof VZDTXK
Digital unterschrieben von Hoppe Frithjof VZDTXK
Datum: 2026.03.20 23:00:02 +01'00'

Name: Benjamin Dätwyler

Unterschrift: Daetwyler Benjamin W2FGMH
20.03.2026
Info: admin.ch/esignature | validator.ch

[3] z.B. APA oder IEEE
[4] z.B. ChatGPT
[5] z.B. Deepl
[6] z.B. Quillbot
[7] z.B. Github Copilot

A Mock-Up zur Veranschaulichung des PoCs

Zur Generierung des Base44 Mock-Up wurde folgender Prompt verwendet:

Design a government cloud governance and compliance portal UI that supports traceability between regulations, governance controls, and cloud resources across multiple cloud platforms.

The portal is used by two main actors: - Compliance Officers - Cloud Experts (Cloud Center of Excellence)

The goal of the portal is to show which regulations are applied to which projects, how they are technically enforced via governance controls and cloud policies, and the current compliance status, based on tags applied to cloud resources.

The UI should include the following features:

1. Global Governance Dashboard - Overview of all regulations, projects, and cloud platforms - High-level compliance status indicators (compliant, at risk, non-compliant) - Filters for cloud platform (AWS, Azure), environment (dev, test, prod), and project

2. Regulation-Centric View - Select a regulation to see: - Regulation description - Mapped governance controls - Affected projects - Compliance status per project and per cloud platform - Visual status indicators (green / yellow / red)

3. Project-Centric View - Select a project to see: - Project metadata (name, owner, environment) - Applied regulations - Associated governance controls - Related cloud resources - Compliance status per regulation

4. Multi-Cloud Platform View - Separate views or tabs for AWS and Azure - Show governed cloud resources per platform - Display applied tags (e.g. regulation ID, data classification, environment) - Show enforcement mechanism (policy, blueprint, landing zone control)

5. Tag-Based Traceability - Visualize traceability from: Regulation → Governance Control → Cloud Policy → Cloud Resource - Use tags on cloud resources as the technical basis for traceability - Clearly show how compliance status is derived from tags and policy evaluations

6. Compliance Status and Evidence - Show current compliance state and last evaluation timestamp - Indicate evidence source (policy evaluation, tag scan) - Designed to support audit and reporting use cases

Visual style: - Clean, professional, government-grade enterprise UI - Focus on clarity, traceability, and transparency - Table-based layouts, filters, status badges - Minimal animations, high readability

Generate a multi-screen UI mock-up including dashboard, regulation detail view, project detail view, and cloud platform views, emphasizing governance, compliance, and traceability.

Der Output ist der folgende

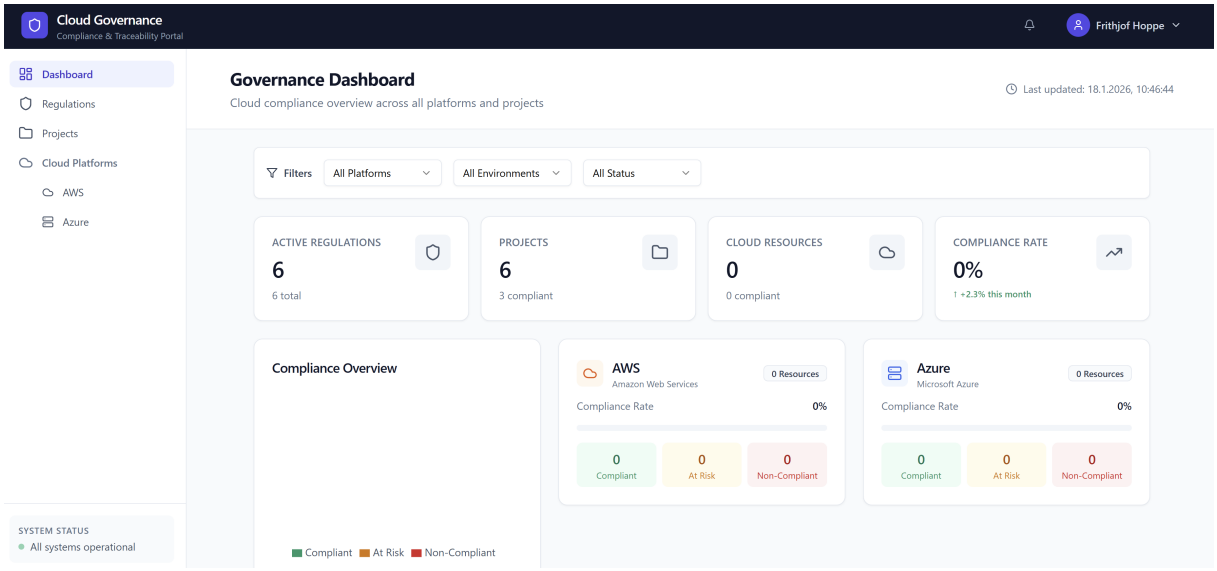


Abbildung A.1: Ansicht Dashboard

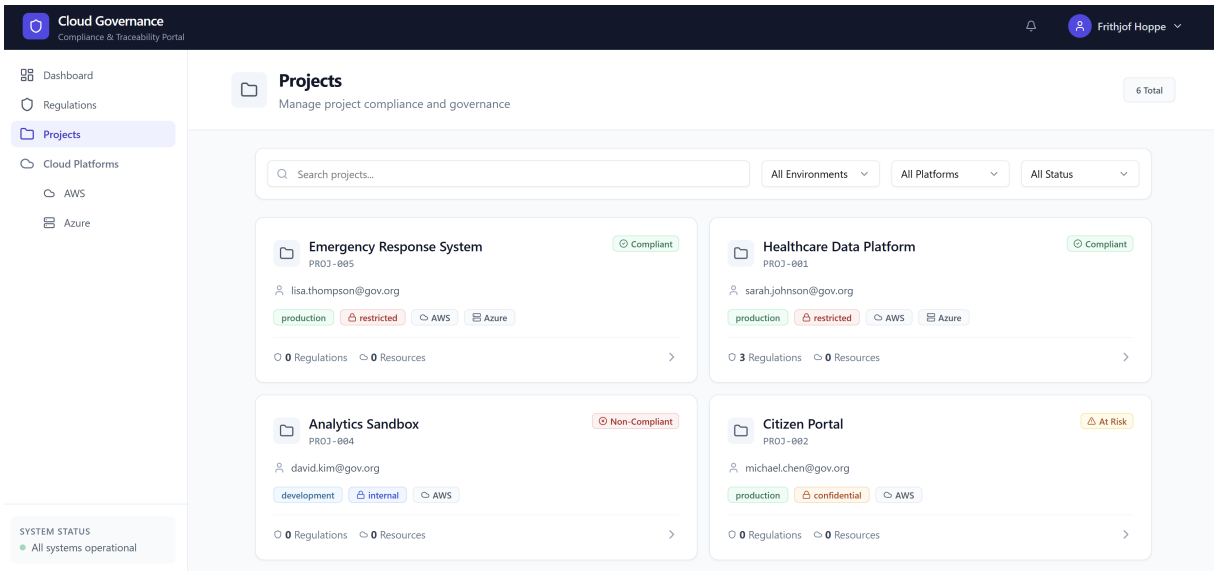


Abbildung A.2: Ansicht Projekte

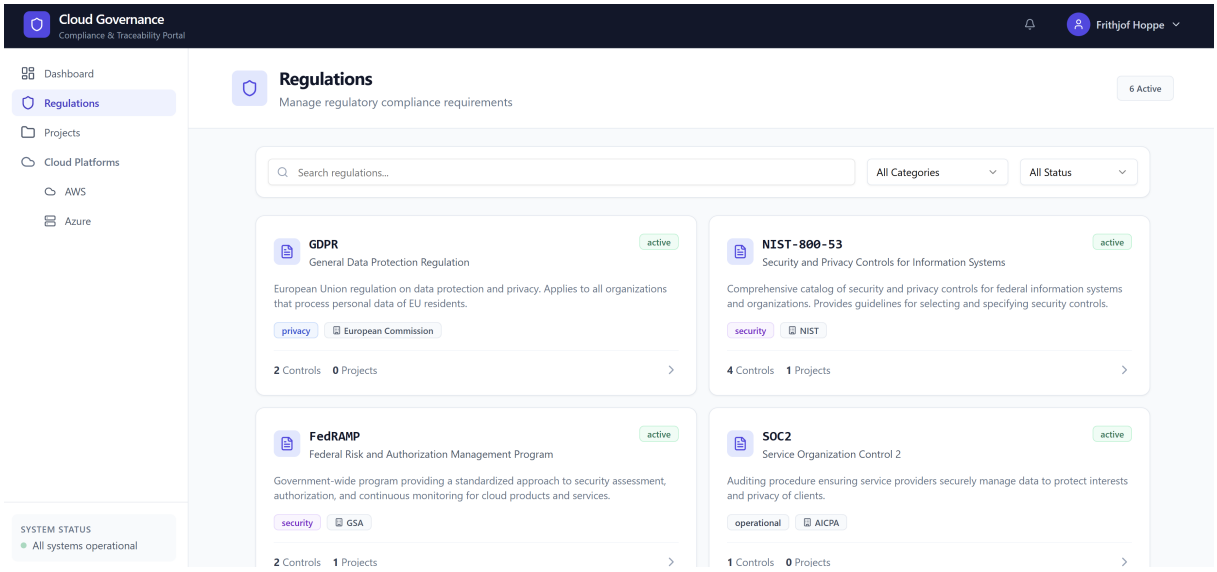


Abbildung A.3: Ansicht Regulations

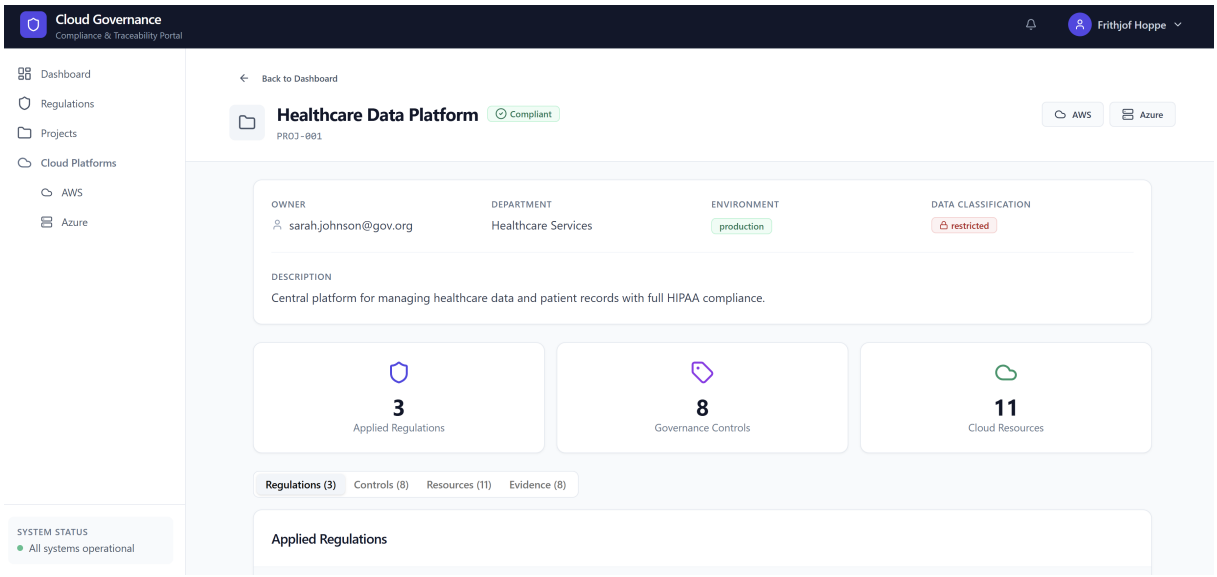


Abbildung A.4: Detailansicht des Beispielprojekts Healthcare

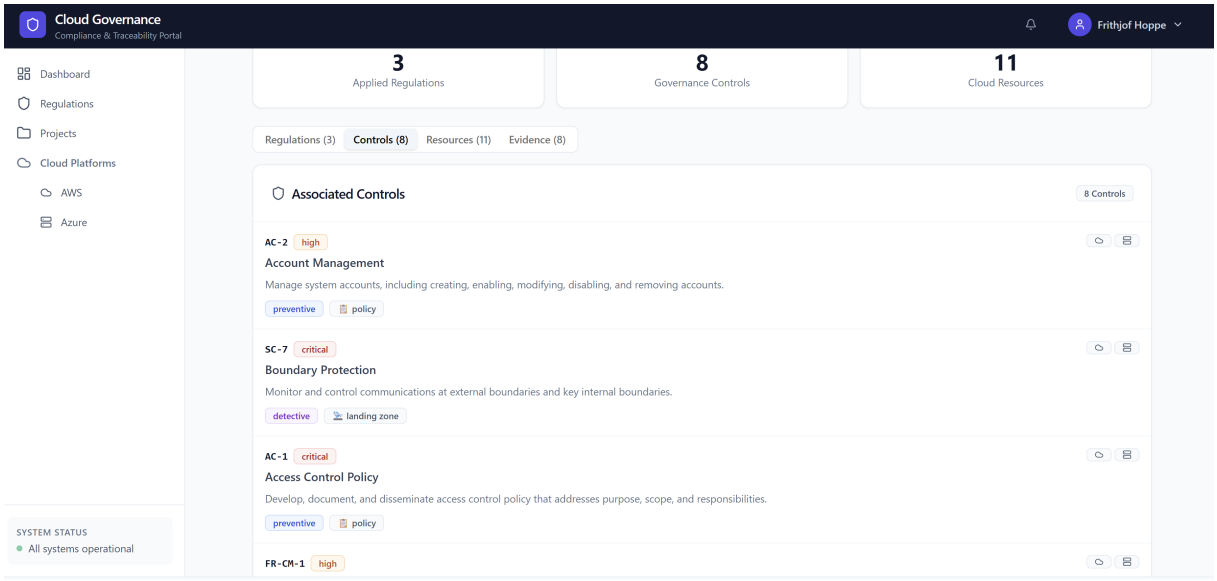


Abbildung A.5: Detailansicht der angewendeten Controls im Beispielprojekt Healthcare

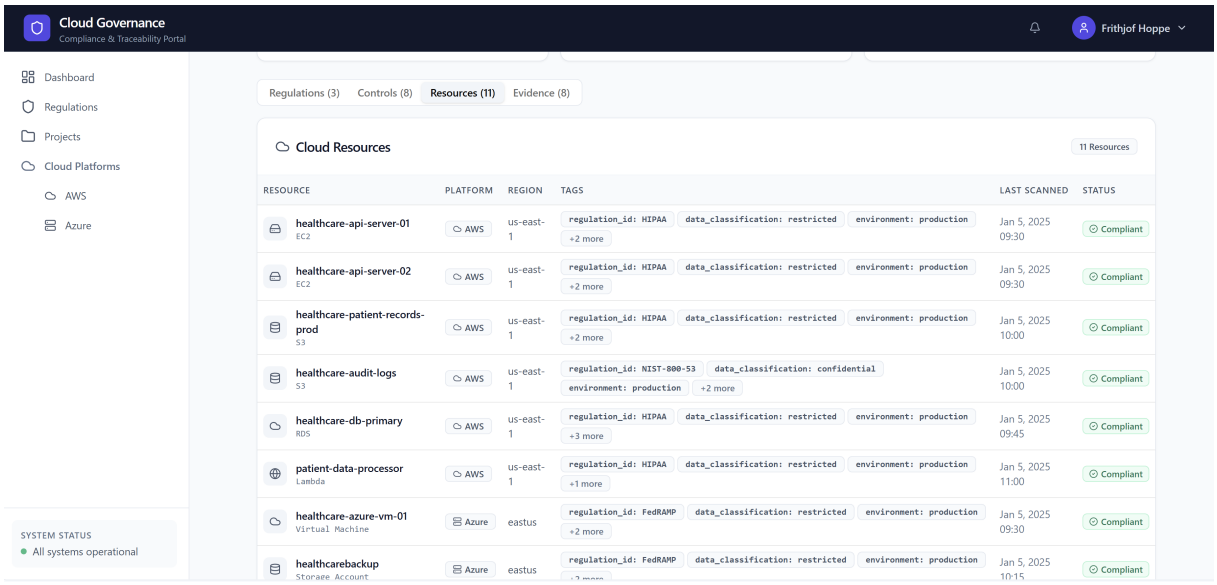


Abbildung A.6: Detailansicht der beinhalteten Resources im Beispielprojekt Healthcare

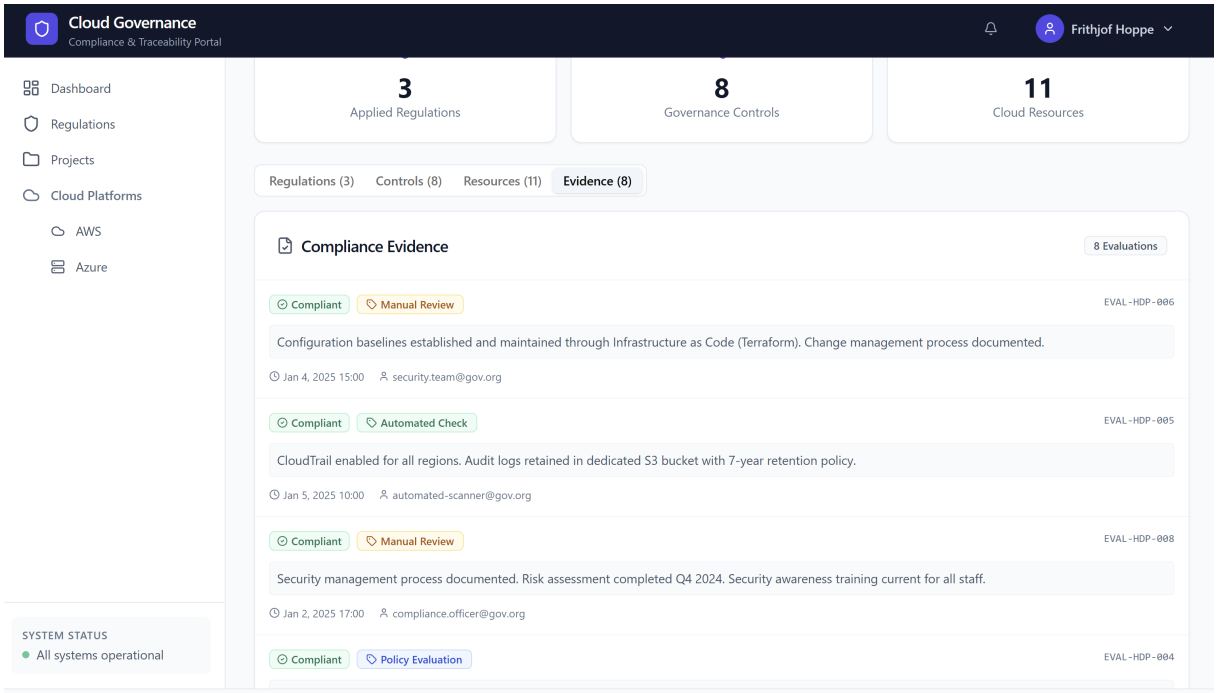


Abbildung A.7: Detailansicht der beinhalteten Evidences im Beispielprojekt Healthcare

B Tool für die Berechnung der Evaluation Scores

Das folgende Python Script wird für die Berechnung der Scores der verschiedenen Traceabilityarchitektur Varianten in Abschnitt 6 verwendet.

```python
# quality_score.py
quality_attributes = {
    "Performance": (0.3, 4),
    "Portability": (1, 1),
    "Auditability": (0.6, 2),
    "Automatability": (1, 4),
    "Maintainability": (0.8, 2),
    "Adaptability": (0.6, 1),
    "Complexity": (0.8, 4),
    "Traceability Strength": (1, 3),
}

weighted_sum = 0
weight_sum = 0

for q, (weight, score) in quality_attributes.items():
    weighted_sum += weight * score
    weight_sum += weight

score_quality = weighted_sum / weight_sum

print("Weighted sum:", weighted_sum)
print("Total weight:", weight_sum)
print("Score_quality:", round(score_quality, 3))
```

Abbildung B.1: Tool zur Berechnung der Scores für die Traceabilityarchitektur Varianten in Unterunterabschnitt 3.5.2

C Report zu Landing Zone Literaturrecherche

Research-Report generiert mit elicit.ai am 08.11.2025 basierend auf der Frage «What are the key architectural components and design principles for establishing a secure and efficient cloud landing zone?»

What are the key architectural components and design principles for establishing a secure and efficient cloud landing zone?

Cloud landing zones are built on seven key architectural components (identity management, network segmentation, encryption, compliance governance, observability, integration, and automation) following modular, standardized designs that prioritize continuous verification and defense in depth strategies.

Abstract

Seven central architectural components underpin a secure and efficient cloud landing zone. Identity and Access Management (IAM) is deemed vital, with studies detailing centralized IAM, Multi-Factor Authentication, and federated identity within a Zero Trust, least-privilege framework. Micro-segmentation and network controls feature prominently via private networks, service meshes (e.g., Istio, Calico), and Layer 2/3 switching to contain potential breaches. Encryption and secrets management are consistently recommended—employing pervasive encryption at rest and in transit, centralized secret stores, and key management—to safeguard data confidentiality.

Additional components include governance and compliance, which rely on configuration baselines, continuous compliance scanning, and audit tools aligned with standards such as CIS, NIST, and ISO; monitoring and observability through centralized logging, distributed tracing, and anomaly detection; integration and connectivity using API gateways and cloud-agnostic overlays; and management and orchestration via automation, DevSecOps practices, and Infrastructure as Code. These studies collectively advocate for modular, standardized, and adaptive designs that emphasize continuous verification, defense in depth, and operational efficiency in hybrid and multi-cloud environments.

Paper search

We performed a semantic search using the query ”What are the key architectural components and design principles for establishing a secure and efficient cloud landing zone?” across over 138 million academic papers from the Elicit search engine, which includes all of Semantic Scholar and OpenAlex.

We retrieved the 50 papers most relevant to the query.

Screening

We screened in sources that met these criteria:

- **Cloud Landing Zone Focus**: Does the study specifically address cloud landing zone design, implementation, or architectural frameworks?
- **Security Aspects**: Does the study include security considerations, controls, or principles related to cloud infrastructure?
- **Efficiency/Performance Focus**: Does the study discuss operational efficiency, resource optimization, scalability, or performance aspects?
- **Major Cloud Platform Coverage**: Does the study cover AWS, Microsoft Azure, Google Cloud Platform, or multi-cloud environments?
- **Evidence-Based Content**: Does the study provide concrete evidence, implementation experiences, technical analysis, or synthesized evidence (rather than being purely theoretical)?
- **Landing Zone Architecture Scope**: Does the study go beyond basic cloud migration to address the specific architectural framework of landing zones?

- **Foundational Infrastructure Focus**: Does the study address foundational infrastructure aspects rather than being limited to single-application deployments?
- **Practical Application**: Does the study include implementation details, practical guidance, or empirical validation (rather than being purely theoretical or conceptual)?
- **Academic Rigor**: Is the study peer-reviewed research rather than an opinion piece, editorial, or non-peer-reviewed content?

We considered all screening questions together and made a holistic judgement about whether to screen in each paper.

Data extraction

We asked a large language model to extract each data column below from each paper. We gave the model the extraction instructions shown below for each column.

- **Architectural Components**:

  Extract all specific technical components that make up the cloud landing zone architecture including:

- Core infrastructure components (networking, compute, storage)
- Security components (identity management, access controls, firewalls, etc.)
- Governance and compliance components
- Monitoring and logging components
- Integration and connectivity components
- Management and orchestration components For each component, note its purpose and role in the overall architecture.

- **Design Principles**:

  Extract all foundational design principles and philosophies that guide the architecture including:

- Security principles (Zero Trust, defense in depth, least privilege, etc.)
- Architectural principles (scalability, modularity, isolation, etc.)
- Operational principles (automation, standardization, etc.)
- Compliance and governance principles Note how each principle influences architectural decisions and component selection.

- **Security Implementation**:

  Extract comprehensive details about security mechanisms and approaches including:

- Authentication and authorization methods
- Encryption implementations (at rest, in transit, in use)
- Network security controls (micro-segmentation, perimeters, etc.)
- Threat detection and response capabilities
- Secrets management approaches
- Compliance and audit controls
- Security monitoring and observability Include specific technologies, standards, or frameworks mentioned.

- **Integration Patterns**:

  Extract information about how components integrate and work together including:

- Inter-component communication patterns

- Integration with existing on-premises systems
- Multi-cloud and hybrid cloud connectivity approaches
- Service mesh and microservices integration
- API and data integration patterns
- Workload migration and portability mechanisms Note any challenges or considerations for integration.

- **Operational Practices**:

  Extract details about deployment, management, and operational aspects including:

- Deployment methodologies and automation approaches
- Configuration management and standardization
- Monitoring, logging, and observability practices
- Incident response and disaster recovery procedures
- Performance optimization and efficiency measures
- Cost management and resource optimization
- DevSecOps and CI/CD integration practices

- **Application Context**:

  Extract information about where and how the landing zone is applied including:

- Target cloud environments (AWS, Azure, GCP, multi-cloud, hybrid)
- Industry sectors and use cases
- Organizational contexts (enterprise size, maturity level)
- Regulatory and compliance requirements addressed
- Scalability and performance requirements
- Geographic and regional considerations Note any limitations or specific applicability constraints.

Results

Characteristics of Included Studies

Study	Study Focus	Cloud Platform	Primary Contribution	Methodology Type	Full text retrieved
Maksimov, 2025	Multi-layered security for cloud infrastructures and web applications	Amazon Web Services (AWS)-like, hybrid, multi-cloud (technology-neutral)	Reproducible architectural and operational framework integrating Zero Trust, defense in depth, observability, and compliance	Conceptual/technical framework (no empirical data)	Yes

Study	Study Focus	Cloud Platform	Primary Contribution	Methodology Type	Full text retrieved
Laheri et al., 2025	Azure Landing Zones for secure, scalable cloud adoption	Microsoft Azure	Architectural principles, best practices, and real-world experience in regulated industries	Abstract-only, conceptual (no empirical data)	No
Bakshi, 2014	Secure hybrid cloud computing approaches and use cases	Hybrid (public/private/community)	Virtual Open Systems Interconnection (OSI) Layer 2 switching, encrypted tunnels, key management, orchestration integration	Technical review (no empirical data)	No
Demchenko et al., 2017	Intercloud Security Framework for multi-cloud, data-intensive applications	Multi-cloud	Federated identity, secure logging, attribute-based access control, compliance	Technical/conceptual (no empirical data)	Yes
Bishukarma, 2023	Scalable Zero-Trust Architectures for multi-cloud Software as a Service (SaaS)	Multi-cloud SaaS	Review of Zero Trust Architecture (ZTA) components (micro-segmentation, Identity and Access Management (IAM), threat detection)	Review paper (no empirical data)	No
Saboowala et al., "Designing Networks and Services for the Cloud"	Network/service design for secure, resilient cloud applications	Enterprise, service provider, hybrid/multi-cloud	Virtualization, segmentation, systematic security, Service Level Agreement (SLA)-driven design	Book/technical guide (no empirical data)	No

Study	Study Focus	Cloud Platform	Primary Contribution	Methodology Type	Full text retrieved
Dasher et al., "Designing Networks and Services for the Cloud"	Security architectures for protecting cloud data planes	Not specified (cloud, on-premises, hybrid)	Security perimeters, landing points, Zero Trust for data plane protection	Abstract-only, conceptual (no empirical data)	No
Ali, 2023	Secure, robust e-commerce platforms for public cloud	Public cloud	Comprehensive design: security, scalability, compliance, DevOps, sustainability	Technical/conceptual (no empirical data)	Yes
Arora and Hastings, 2024	Microsegmented, zero trust multi-cloud network architecture	Multi-cloud	Open-source tools (Istio, Calico, Kubernetes), operational controls, layered design	Technical/conceptual (no empirical data)	Yes
Adanigbo et al., 2024	Zero Trust Security in multi-cloud microservices platforms	Multi-cloud	Identity-centric access, policy engines, micro-segmentation, service mesh	Review/conceptual framework (no empirical data)	No

Summary of Study Characteristics:

- Cloud Platform:
  - Three studies focused on multi-cloud environments.
  - One study addressed multi-cloud Software as a Service (SaaS) specifically.
  - One study focused on hybrid (public/private/community) cloud.
  - One study focused on a single public cloud (Microsoft Azure).
  - One study addressed public cloud (not Azure).
  - One study was technology-neutral (Amazon Web Services-like, hybrid, multi-cloud).
  - One study addressed enterprise, service provider, and hybrid/multi-cloud collectively.
  - One study did not specify the cloud platform.
- Primary Contribution:
  - Three studies contributed architectural frameworks or comprehensive designs for cloud security.
  - One study provided architectural principles and best practices.
  - Three studies focused on operational or technical controls and tools.
  - Two studies reviewed components or technologies (for example, Zero Trust, micro-segmentation).
  - One study was a book or technical guide.

- Methodology Type:
  - Four studies used a conceptual or technical framework approach (no empirical data).
  - Two studies were abstract-only or conceptual (no empirical data).
  - One study was a technical review (no empirical data).
  - Two studies were review papers (no empirical data).
  - One study was a book or technical guide (no empirical data).
- Evidence Base:
  - All included studies are conceptual, technical, review, or guide-based in nature. We did not find any studies using empirical or experimental methodologies. For studies where only the abstract was available, findings are limited in detail and should be interpreted with caution.

Thematic Analysis

Security Architecture and Zero Trust Implementation

- Zero Trust and Multi-layered Security:Most included studies describe Zero Trust principles and multi-layered security architectures as central, based on conceptual frameworks. Maksimov (2025) and Arora and Hastings (2024) detail architectures that integrate Zero Trust, defense in depth, least privilege, and pervasive encryption, with micro-segmentation and identity-centric access controls as foundational elements.
- Continuous Verification and Policy Engines:Adanigbo et al. (2024) and Bishukarma (2023) emphasize continuous verification, centralized policy engines, and service mesh integration for context-aware access enforcement.
- Data Protection and Compliance:Ali (2023) and Saboowala et al. focus on data protection, compliance, and systematic security, while Bakshi (2014) and Demchenko et al. (2017) highlight federated identity, key management, and secure communication in hybrid and multi-cloud contexts.
- Evidence Base:The included studies consistently advocate for Zero Trust and layered security, though these recommendations are based on conceptual or review literature rather than empirical validation.

Network Architecture and Micro-segmentation

- Micro-segmentation as a Core Control:Maksimov (2025), Arora and Hastings (2024), and Bishukarma (2023) describe micro-segmentation as a core security control, implemented via private networks, service mesh (such as Istio), and network policy engines.
- Virtualized Network Overlays:Bakshi (2014) and Saboowala et al. discuss virtualized network overlays and Open Systems Interconnection (OSI) Layer 2/3 switching for hybrid and multi-cloud connectivity.
- Cloud Security Perimeters:Dasher et al. (2022) introduces the concept of Cloud Security Perimeters.
- Technical Evolution:More recent studies (Maksimov, 2025; Arora and Hastings, 2024; Bishukarma, 2023) leverage open-source tools and cloud-native constructs, while older works (Bakshi, 2014; Saboowala et al.) focus on virtualization and network overlays.

Governance Framework and Compliance Integration

- Alignment with Standards:Maksimov (2025) and Demchenko et al. (2017) align their frameworks with international standards such as the Center for Internet Security (CIS), National Institute of Standards and Technology

(NIST), International Organization for Standardization (ISO), and Cloud Security Alliance (CSA), emphasizing standardized configuration baselines, continuous compliance scanning, and adaptive policy enforcement.
- Data Sovereignty and Audits:Ali (2023) and Saboowala et al. address data sovereignty, regular audits, and legal considerations, particularly for regulated industries.
- Governance Tools:Adanigbo et al. (2024) and Arora and Hastings (2024) discuss governance tools and audit mechanisms as part of the management layer.
- Consistency and Detail:The focus on compliance is consistent across studies, though the level of operational detail and empirical validation varies.

Operational Excellence and Automation

- Automation and Standardization:Automation, standardization, and DevSecOps (Development, Security, and Operations) practices are widely advocated for efficient, resilient cloud landing zones.
- Automated Policy Enforcement:Maksimov (2025) details automated policy enforcement, centralized secrets management, and security gates in Continuous Integration/Continuous Deployment (CI/CD) pipelines.
- Infrastructure as Code and Monitoring:Ali (2023) and Saboowala et al. highlight CI/CD, Infrastructure as Code (IaC), and proactive monitoring as essential for operational excellence.
- Open-source Management Tools:Arora and Hastings (2024) leverage Kubernetes-based microservices and open-source management tools for flexible, cost-effective operations.
- Integration with Orchestration:Demchenko et al. (2017) and Bakshi (2014) discuss integration with orchestration and automation platforms, though with less focus on cloud-native toolchains.
- Descriptive Evidence:The evidence base is descriptive, with best practices converging on automation and continuous improvement.

Scalability and Multi-Cloud Considerations

- Adaptability and Modularity:Maksimov (2025), Bishukarma (2023), and Adanigbo et al. (2024) emphasize frameworks that are adaptable to hybrid and multi-cloud environments, supporting secure workload mobility and dynamic service discovery.
- Multi-cloud Connectivity and Federation:Arora and Hastings (2024) and Demchenko et al. (2017) describe architectures that facilitate multi-cloud connectivity and federation.
- Hybrid Cloud and Network Overlays:Bakshi (2014) and Saboowala et al. focus on hybrid cloud and network overlays.
- Resilience and Performance:Ali (2023) addresses multi-regional deployment and disaster recovery for e-commerce, highlighting the need for resilience and performance at scale.
- Advocacy for Scalable Architectures:The literature consistently advocates for scalable, modular architectures, though empirical validation of scalability claims is limited.

Synthesis of Key Components and Principles

Architectural Component	Design Principles	Implementation Approaches	Efficiency Considerations
Identity and Access Management (IAM)	Zero Trust, least privilege, continuous verification	Centralized IAM, Multi-Factor Authentication (MFA), short-lived credentials, federated identity, policy engines	Reduces risk, supports compliance, enables automation
Micro-segmentation & Network Controls	Defense in depth, isolation, modularity	Private networks, service mesh (Istio), Layer 2/3 switching, Calico, Cloud Security Perimeters	Limits lateral movement, granular policy enforcement, supports multi-cloud
Encryption & Secrets Management	Pervasive encryption, centralized secrets, key management	Encryption at rest/in transit (Transport Layer Security (TLS), Mutual TLS (mTLS)), centralized secret stores, runtime injection	Ensures data confidentiality, supports compliance, operational efficiency
Governance & Compliance	Standardization, adaptive policy, auditability	Configuration baselines, compliance scanning, audit tools, alignment with CIS/NIST/ISO	Maintains regulatory alignment, reduces drift, supports audits
Monitoring & Observability	Total observability, incident response, continuous improvement	Centralized logging, distributed tracing, Security Information and Event Management (SIEM), anomaly detection	Improves Mean Time to Detect/Mean Time to Respond (MTTD/MTTR), supports incident response, operational resilience
Integration & Connectivity	Cloud-agnostic, modularity, interoperability	Application Programming Interface (API) gateways, service mesh, hybrid/multi-cloud overlays, workload migration	Enables extensibility, workload mobility, supports multi-cloud
Management & Orchestration	Automation, standardization, DevSecOps	Continuous Integration/Continuous Deployment (CI/CD), Infrastructure as Code (IaC), automated remediation, configuration management, open-source tools	Reduces manual effort, increases consistency, cost-effective operations

Summary of Key Findings:

- Design Principles:
  - Modularity and standardization were the most frequently mentioned design principles (each in two architectural components).
  - Other principles—Zero Trust, least privilege, continuous verification, defense in depth, isolation, pervasive encryption, centralized secrets, key management, adaptive policy, auditability, total observability, incident response, continuous improvement, cloud-agnostic, interoperability, automation, and DevSecOps—were each mentioned in one component.
- Implementation Approaches:
  - Service mesh was mentioned in two components.
  - All other approaches—including centralized IAM, Multi-Factor Authentication, short-lived credentials, federated identity, policy engines, private networks, Layer 2/3 switching, Calico, Cloud Security Perimeters, encryption at rest/in transit, Transport Layer Security/Mutual TLS, centralized secret stores, runtime injection, configuration baselines, compliance scanning, audit tools, alignment with CIS/NIST/ISO, centralized logging, distributed tracing, Security Information and Event Management, anomaly detection, Application Programming Interface gateways, hybrid/multi-cloud overlays, workload migration, Continuous Integration/Continuous Deployment, Infrastructure as Code, automated remediation, configuration management, and open-source tools—were each mentioned in one component.
- Efficiency Considerations:
  - "Supports compliance" and "supports multi-cloud" were each mentioned in two components.
  - All other efficiency considerations—including reduces risk, enables automation, limits lateral movement, granular policy enforcement, ensures data confidentiality, operational efficiency, maintains regulatory alignment, reduces drift, supports audits, improves Mean Time to Detect/Mean Time to Respond, supports incident response, operational resilience, enables extensibility, workload mobility, reduces manual effort, increases consistency, and cost-effective operations—were each mentioned in one component.
- Completeness of Data:
  - All selected columns for the seven architectural components were populated based on the included studies.

: Maksimov, 2025 : Bakshi, 2014 : Demchenko et al., 2017 : Bishukarma, 2023 : Saboowala et al. : Dasher et al., 2022 : Ali, 2023 : Arora and Hastings, 2024 : Adanigbo et al., 2024

References

Grant Dasher, Ines Envid, and B. Calder. "Architectures for Protecting Cloud Data Planes." *arXiv.org*, 2022.

Huseni Saboowala, M. Abid, and Sudhir Modali. "Designing Networks and Services for the Cloud: Delivering Business-Grade Cloud Applications and Services," 2013.

Kapil Bakshi. "Secure Hybrid Cloud Computing: Approaches and Use Cases." *IEEE Aerospace Conference*, 2014.

Oluwasanmi Segun Adanigbo, Bolaji Iyanu Adekunle, Ejielo Ogbuefi, Oyejide Timothy Odofin, Oluwademilade Aderemi Agboola, and Denis Kisina. "Implementing Zero Trust Security in Multi-Cloud Microservices Platforms: A Review and Architectural Framework." *International Journal of Advanced Multidisciplinary Research and Studies*, 2024.

Ramesh Bishukarma. "Scalable Zero-Trust Architectures for Enhancing Security in Multi-Cloud SaaS Platforms." *International Journal of Advanced Research in Science, Communication and Technology*, 2023.

Rohit Laheri, Harish kumar Krishnamurthy Sukumar, Krishnamurthy, Sukumar, Chandrashekar Kola, and Yashasvi Makin. "Designing Secure and Scalable Cloud Infrastructures Using Azure Landing Zones." *Journal of Information Systems Engineering & Management*, 2025.

Sunil Arora, and John D. Hastings. "Microsegmented Cloud Network Architecture Using Open-Source Tools for a Zero Trust Foundation." *International Conference on Security of Information and Networks*, 2024.

Syed Afraz Ali. "Designing Secure and Robust E-Commerce Plaform for Public Cloud." *The Asian Bulletin of Big Data Management*, 2023.

Vladyslav Maksimov. "ARCHITECTURAL PRINCIPLES AND OPERATIONAL PRACTICES FOR BUILDING SECURE DIGITAL INFRASTRUCTURE IN CLOUD ENVIRONMENTS." *Terra Security*, 2025.

Y. Demchenko, F. Turkmen, C. D. Laat, and Mathias Slawik. "Defining Intercloud Security Framework and Architecture Components for Multi-Cloud Data Intensive Applications." *IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing*, 2017.

D Report zu multi-cloud Architecture-Principles Literaturrecherche v1

Research-Report generiert mit elicit.ai am 30.12.2025 basierend auf der Frage «Which normative, technology-agnostic enterprise architecture principles must hold so that multiple cloud-specific solution architectures can be derived consistently?»

Which normative, technology-agnostic enterprise architecture principles must hold so that multiple cloud-specific solution architectures can be derived consistently?

Five normative, technology-agnostic enterprise architecture principles must hold for consistent derivation of cloud-specific solution architectures: architectural abstraction for portability, modular decomposition of cloud stacks, explicit reference models for building blocks, governance primacy through well-defined business architecture, and security reconfigurability across environments, with all five principles requiring orchestrated implementation rather than individual application.

Abstract

This systematic review of 10 sources identifies five core normative, technology-agnostic enterprise architecture principles necessary for consistent derivation of cloud-specific solution architectures: architectural abstraction to enable portability through containerization and platform-agnostic interfaces ; modular decomposition of cloud stacks into independently selectable components ; explicit reference models providing taxonomy for architecture building blocks and relationships ; governance primacy requiring well-defined business architecture and policies as preconditions ; and security reconfigurability enabling algorithmic policy transformation across environments . However, the evidence reveals that no single principle suffices—consistent derivation requires orchestrated implementation of all five principles, with sequencing determined by organizational maturity and architectural scope. Studies demonstrating successful implementation restricted scope to bounded hybrid architectures with explicit validation mechanisms , while those addressing enterprise-level multi-cloud scenarios identified substantial barriers including operational complexity, governance inconsistencies, and technical compatibility challenges . The evidence base remains limited, with five of ten sources available only as abstracts and validation primarily through case studies or theoretical justification rather than empirical testing across diverse organizational contexts. Organizations lacking foundational governance and assessment instruments cannot effectively implement advanced abstraction principles, suggesting that principle applicability depends critically on organizational maturity level rather than representing universal requirements.

Paper search

We performed a semantic search using the query ”Which normative, technology-agnostic enterprise architecture principles must hold so that multiple cloud-specific solution architectures can be derived consistently?” across over 138 million academic papers from the Elicit search engine, which includes all of Semantic Scholar and OpenAlex.

We retrieved the 50 papers most relevant to the query.

Screening

We screened in sources based on their abstracts that met these criteria:

- **Enterprise Architecture Principles**: Does the study address normative principles or frameworks that guide architectural decisions at the enterprise level?
- **Cloud Computing Context**: Does the research involve cloud computing architectures or multi-cloud environments?
- **Principle-Implementation Relationship**: Does the study examine or demonstrate how high-level architectural principles translate into specific solution implementations?

- **Study Type and Context**: Is the study an empirical study, case study, systematic review, meta-analysis, or theoretical framework conducted in an enterprise or organizational context?
- **Beyond Technical Implementation Only**: Does the study go beyond focusing solely on technical implementation details (coding, deployment, operations) to include architectural principles or higher-level architectural guidance?
- **Broader Architectural Context**: Does the research extend beyond single-cloud vendor solutions to include broader architectural context or technology-agnostic principles?
- **Normative or Prescriptive Elements**: Does the study include normative or prescriptive elements (principles, frameworks, guidance for future decisions) rather than being purely descriptive of existing systems?

We considered all screening questions together and made a holistic judgement about whether to screen in each paper.

Data extraction

We asked a large language model to extract each data column below from each paper. We gave the model the extraction instructions shown below for each column.

- **EA Principles**:

  Extract all enterprise architecture principles explicitly mentioned or proposed in the study, including:

- Name/title of each principle
- Full description or definition provided
- Whether characterized as normative (prescriptive) or descriptive
- Technology-agnostic vs technology-specific classification
- Source or authority cited (e.g., TOGAF, proprietary framework, derived from analysis)

- **Multi-Cloud Relevance**:

  Extract evidence of how the principles relate to multi-cloud or vendor-agnostic approaches:

- Explicit discussion of multi-cloud scenarios
- Vendor lock-in prevention mechanisms
- Technology portability considerations
- Cross-platform consistency requirements
- Platform abstraction strategies mentioned

- **Derivation Process**:

  Extract any described methods, processes, or mechanisms for deriving specific cloud architectures from general principles:

- Step-by-step processes described
- Decision frameworks or models presented
- Transformation rules or guidelines
- Consistency mechanisms or validation approaches
- Reference models or templates provided

- **Supporting Evidence**:

  Extract the type and quality of evidence supporting the identified principles:

- Empirical validation (case studies, experiments, surveys)

- Theoretical justification or reasoning provided
- Industry examples or practical applications cited
- Expert opinion or consensus mentioned
- Literature review findings that support principles

- **Implementation Challenges**:

  Extract identified challenges, constraints, or prerequisites for applying technology-agnostic EA principles:

- Technical barriers to implementation
- Organizational or governance challenges
- Skills or capability requirements
- Trade-offs between agnosticism and optimization
- Success factors or prerequisites mentioned

- **Scope Context**:

  Extract contextual information about the scope and applicability of findings:

- Industry sectors or domains addressed
- Organization types or sizes considered
- Cloud service models covered (IaaS, PaaS, SaaS)
- Technology maturity levels discussed
- Geographic or regulatory context mentioned

Results

Characteristics of Included Studies

All ten sources addressed enterprise architecture principles in cloud computing contexts, though with varying degrees of specificity and full-text availability. The table below summarizes key characteristics of each source.

Study	Full text retrieved?	Primary focus	Framework basis	Cloud service models addressed
Emmanouil D. Tritsiniotis et al., 2016	No	Cloud-enabled EA framework development	TOGAF and ArchiMate	Cloud ecosystems broadly
Srikanth Gurram et al., 2025	No	Multi-cloud architecture principles	Not specified	Multi-cloud infrastructure
Ovidiu Noran et al., 2017	No	Business cloudification readiness	Enterprise Architecture-based approach	Broad cloud applicability
Mohammad Y. Hajjat et al., 2010	Yes	Hybrid cloud migration planning	Proprietary model	IaaS
Thomas Erl et al., 2013	No	Cloud computing concepts and architecture	Vendor-neutral	IaaS, PaaS, SaaS

Study	Full text retrieved?	Primary focus	Framework basis	Cloud service models addressed
Nane Kratzke et al., 2016	No	Cloud-native application reference model	Standardized IaaS services	IaaS
Liang-Jie Zhang et al., 2009	Yes	Cloud Computing Open Architecture (CCOA)	SOA and virtualization integration	IaaS, PaaS, SaaS
M. Papazoglou et al., 2012	No	Cloud federation integration	Modular cloud delivery model	SaaS, PaaS, IaaS
Izabelle Hannemann et al., 2022	No	Digital transformation technology influence on EA	TOGAF principles	Cloud computing broadly
L. Maciaszek et al., 2014	No	Meta-architecture for service cloud applications	PCBMER, SANTA, MAAG	Service cloud applications

Five of the ten sources had only abstracts available, limiting the depth of principle extraction possible. Studies with full text (Mohammad Y. Hajjat et al., 2010 and Liang-Jie Zhang et al., 2009) provided more detailed technical specifications and validation approaches. The majority of sources focused on enterprise-level organizations with broad industry applicability , though some addressed specific sectors such as insurance and regulated industries .

Framework foundations varied considerably, with TOGAF serving as the basis for multiple approaches , while others proposed proprietary models or integrations of existing frameworks . Cloud service model coverage spanned the full spectrum from IaaS-focused approaches to comprehensive treatments of IaaS, PaaS, and SaaS .

Thematic Analysis

Technology Abstraction and Vendor Neutrality

The most consistently emphasized normative principle across sources was architectural abstraction to enable portability and prevent vendor lock-in. Srikanth Gurram et al. (2025) explicitly identified architectural abstraction as a normative, technology-agnostic principle for achieving portability , supported by containerization technologies and platform-agnostic interfaces . This principle manifested through multiple mechanisms: cloud management platforms, unified identity frameworks, data orchestration tools, and software-defined networking to create integration frameworks for coherent operations .

Thomas Erl et al. (2013) reinforced this theme through vendor-neutral coverage, emphasizing technology portability considerations and cross-platform consistency requirements . The practical implementation of vendor neutrality appeared in Nane Kratzke et al. (2016), which presented a reference model relying exclusively on well-standardized IaaS services to prevent vendor lock-in . Similarly, Liang-Jie Zhang et al. (2009) emphasized the need to eliminate provider-specific dependencies in PaaS and IaaS environments, proposing a generic security framework capable of interfacing with any cloud environment to ensure cross-platform consistency .

M. Papazoglou et al. (2012) framed the vendor lock-in problem as stemming from rigid, monolithic cloud stacks that prevent consumers from dynamically mixing and matching functionality across different tiers . Their proposed solu-

tion involved transforming the cloud stack into modular, easily combined components through an enhanced cloud delivery model . This modularity principle enables organizations to maintain flexibility while avoiding dependencies on specific vendor implementations.

However, Mohammad Y. Hajjat et al. (2010) acknowledged vendor lock-in as an ongoing challenge without providing detailed prevention mechanisms, though their future directions suggested generalizing models to support multiple cloud locations . The theoretical basis for vendor neutrality appeared consistently across sources , though practical validation remained limited primarily to case studies .

Architectural Reference Models and Derivation Frameworks

Multiple sources proposed structured frameworks for deriving cloud-specific architectures from general principles, though approaches varied substantially in specificity and completeness. Emmanouil D. Tritsiniotis et al. (2016) developed "TOGAF for Cloud Ecosystems" as a cloud-enabled version of TOGAF, providing two key reference models: the Cloud Ecosystem Reference Model, which offers taxonomy for architecture building blocks, organizational roles, and services; and the Enterprise Ecosystem Model, which outlines relationships and dependencies between enterprise frameworks for managing cloud service lifecycles . This approach included specific guidelines for every phase of the TOGAF Architecture Development Method (ADM) , demonstrated through the ArchiSurance case study and validated by expert interviews from multinational companies .

Nane Kratzke et al. (2016) presented a reference model explicitly designed to guide technology identification, classification, adoption, research, and development processes for cloud-native applications . By restricting the model to standardized IaaS services, they aimed to create a foundation for consistent derivation while avoiding vendor-specific dependencies . This theoretical approach prioritized standardization as the mechanism for ensuring cross-platform consistency.

Liang-Jie Zhang et al. (2009) proposed the Cloud Computing Open Architecture (CCOA) based on seven architectural principles that bridge SOA and virtualization . Though the specific principles were not detailed in the available abstract , the approach was validated through two case studies on Infrastructure and Business Cloud , demonstrating practical application. Their security management models and generic security framework provided mechanisms for deriving secure architectures across different cloud environments .

Mohammad Y. Hajjat et al. (2010) developed a more procedural approach through a model for planning hybrid cloud migrations. This model systematically determined which application components to migrate by considering enterprise policies, cost savings, and potential increases in transaction delays and communication costs . The model used graph structures to represent component interactions and included constraints for mean delay, variance, and percentiles of transaction delays to ensure performance targets . Algorithms for reconfiguring reachability policies during migration ensured correctness and minimized unwanted Internet traffic , providing a validation mechanism for architectural consistency. This approach received empirical validation through real enterprise applications and Azure-based deployments .

Thomas Erl et al. (2013) provided decision frameworks through templates and formulas for calculating SLA-related quality-of-service values, along with 29 architectural models and 20 mechanisms . While comprehensive, the abstract did not specify how these models enable consistent derivation across multiple cloud platforms.

Well-Defined Business Architecture and Governance

Several sources emphasized the foundational necessity of well-defined business architecture, policies, and principles as prerequisites for consistent cloud architecture derivation. Ovidiu Noran et al. (2017) explicitly stated that well-

defined business architecture, policies, and principles dictating solution selection and design constitute sine qua non preconditions for successful cloudification . Their Enterprise Architecture-based approach enabled informed decision-making on cloudification extent, type, and provider selection , though specific principles were not detailed .

Emmanouil D. Tritsiniotis et al. (2016) highlighted how cloud computing transforms business goals, operations, service offerings, processes, partnerships, and IT infrastructure—all elements requiring explicit treatment in Enterprise Architecture . The lack of cloud-enabled EA frameworks prior to their proposed approach represented a significant technical barrier , suggesting that explicit framework support is necessary for consistent derivation.

Izabelle Hannemann et al. (2022) evaluated the influence of digital transformation technologies on TOGAF's 21 EA principles , identifying Big Data and Cloud Computing as having the greatest effect on analyzed principles . However, the lack of adequate instruments for assessing current state and identifying improvement opportunities represented a significant technical barrier , indicating gaps in practical tools for applying normative principles consistently.

The importance of governance emerged across multiple sources. Srikanth Gurram et al. (2025) identified governance inconsistencies as a key implementation challenge , while M. Papazoglou et al. (2012) noted that monolithic cloud stacks and one-size-fits-all mentalities create organizational and governance challenges . Liang-Jie Zhang et al. (2009) highlighted security concerns and the risk of outsourced services going out of control in hybrid environments as governance challenges requiring holistic security approaches .

Security, Integration, and Operational Principles

Security and integration principles emerged as critical enablers for consistent multi-cloud architectures, though implementation challenges remained substantial. Liang-Jie Zhang et al. (2009) identified security as a fundamental factor for success, emphasizing the need for holistic security approaches . They proposed a generic security framework for optimized cost-performance ratios , validated through case studies and supported by literature review findings . Security complexity in public clouds compared to private clouds, lack of transparency about data storage locations, and encryption and key management challenges represented significant technical barriers .

Mohammad Y. Hajjat et al. (2010) developed algorithms for ensuring assurable reconfiguration of security policies during cloud migration, including inference of original reachability policies, transformation for new cloud environments, and correct placement of access control lists . Their approach addressed industry-specific regulations and national privacy laws as organizational challenges , demonstrating practical considerations for regulated industries .

M. Papazoglou et al. (2012) proposed integration-as-a-service functionality as a mechanism for enabling cross-platform consistency . Their enhanced cloud delivery model aimed to overcome the rigidity of contemporary cloud stacks through modular, easily combined components , though only theoretical justification was provided .

Operational principles manifested through considerations of performance, scalability, and complexity. Srikanth Gurram et al. (2025) identified operational complexity and technical compatibility as key technical barriers , while Mohammad Y. Hajjat et al. (2010) highlighted performance requirements, network latency, and scalability issues as constraints . Financial management emerged as both a capability requirement and a consideration in migration planning , indicating that economic principles must complement technical ones for consistent architecture derivation.

Synthesis

The literature reveals a fundamental tension between the need for technology-agnostic principles that enable consistent derivation across multiple cloud platforms and the practical challenges of implementing such principles in

heterogeneous cloud environments. This tension manifests differently across three primary contexts: framework development level, implementation complexity level, and organizational maturity level.

At the framework development level, sources can be categorized by their approach to abstraction. Studies proposing reference models based on standardized services (Nane Kratzke et al., 2016 ; Liang-Jie Zhang et al., 2009) prioritized vendor lock-in prevention through deliberate restriction to well-standardized IaaS services and generic security frameworks . These approaches achieved technology-agnosticism through service standardization but provided limited guidance on handling proprietary cloud capabilities. In contrast, studies focusing on enhanced delivery models (M. Papazoglou et al., 2012 ; Emmanouil D. Tritsiniotis et al., 2016) emphasized modular decomposition of cloud stacks into easily combined components and comprehensive phase-by-phase ADM guidelines . These approaches offered more granular derivation mechanisms but required greater organizational investment in framework adoption and tool development.

Implementation complexity level explains apparent contradictions in practical applicability. Studies finding high implementation barriers (Srikanth Gurram et al., 2025 ; Liang-Jie Zhang et al., 2009 ; M. Papazoglou et al., 2012) consistently identified operational complexity, governance inconsistencies, and technical compatibility challenges alongside security concerns, distributed attack risks, and lack of SOA adoption . These sources addressed enterprise-level organizations managing multiple cloud providers simultaneously, where integration complexity scales non-linearly with the number of platforms. Conversely, studies demonstrating successful implementation (Mohammad Y. Hajjat et al., 2010 ; Emmanouil D. Tritsiniotis et al., 2016) focused on hybrid architectures with bounded scope—either planning migration of specific application components or demonstrating principles through focused case studies like ArchiSurance . Both successful approaches included explicit validation mechanisms: performance constraint checking and security policy reconfiguration algorithms or expert evaluation from multinational companies . This suggests that technology-agnostic principles become practically implementable when scope is deliberately bounded and validation mechanisms are explicitly defined.

Organizational maturity level determines which principles prove actionable. Studies emphasizing prerequisites (Ovidiu Noran et al., 2017 ; Izabelle Hannemann et al., 2022 ; Emmanouil D. Tritsiniotis et al., 2016) identified well-defined business architecture, policies, and principles as sine qua non preconditions and noted that lack of adequate assessment instruments leaves organizations uncertain about improvement opportunities . Organizations lacking these foundational elements encounter transformational challenges when cloud computing alters business goals, operations, and IT infrastructure . Studies proposing advanced abstractions (Srikanth Gurram et al., 2025 ; Thomas Erl et al., 2013) assume organizational capability to implement containerization technologies, platform-agnostic interfaces, and unified identity frameworks or to utilize complex templates, formulas, and 29 architectural models . The maturity gap explains why theoretically sound principles achieve limited practical adoption: organizations must first establish governance foundations before leveraging advanced abstraction mechanisms.

Several key normative, technology-agnostic principles emerge as necessary conditions for consistent derivation across these contexts:

Architectural Abstraction Principle : Applications must be designed using abstraction layers that separate business logic from cloud-specific implementations . This enables portability through containerization technologies and platform-agnostic interfaces while supporting vendor-neutral architecture coverage . Validation through standardized IaaS services and generic frameworks demonstrates practical applicability.

Modular Decomposition Principle : Cloud architectures must decompose monolithic stacks into modular, easily combined components that can be independently selected, integrated, and replaced . This principle enables mixing and matching functionality across cloud tiers while preventing rigidity that leads to vendor lock-in .

Reference Model Principle : Consistent derivation requires explicit reference models providing taxonomy for archi-

tecture building blocks, organizational roles, services, and their relationships . These models form common languages for architectural integrity and guide technology identification, classification, and adoption processes .

Governance Primacy Principle : Well-defined business architecture, policies, and principles must dictate solution selection and design as preconditions for cloudification . Governance frameworks must address security as a fundamental success factor through holistic approaches managing complexity, risk, and compliance .

Security Reconfigurability Principle : Security policies must be explicitly modeled and algorithmically transformable during cloud migration to ensure correctness across environments . This requires generic security frameworks capable of interfacing with any cloud environment while maintaining assurable policy enforcement .

For regulated industries and organizations managing multiple cloud platforms simultaneously, performance constraint validation and systematic component placement planning become additional necessary principles. Organizations in early cloud adoption stages should prioritize governance establishment and assessment instrument development before attempting advanced multi-cloud abstractions. Organizations with mature EA practices can leverage comprehensive frameworks with phase-specific guidelines and extensive architectural models to derive consistent architectures across diverse cloud platforms.

The synthesis reveals that no single principle suffices; rather, consistent derivation requires orchestration of abstraction, modularity, reference models, governance, and security reconfigurability principles, with implementation sequencing determined by organizational maturity and architectural scope. Validation mechanisms—whether algorithmic , case-study-based , or expert-evaluated —prove essential for confirming that derived architectures maintain consistency across cloud platforms while satisfying enterprise-specific constraints.

References

Emmanouil D. Tritsiniotis. "Get Ready for the Cloud: Tailoring Enterprise Architecture for Cloud Ecosystems," 2016.

Izabelle Hannemann, Sarah Rodrigues, E. Loures, F. Deschamps, and J. Cestari. "Applying a Decision Model Based on Multiple Criteria Decision Making Methods to Evaluate the Influence of Digital Transformation Technologies on Enterprise Architecture Principles." *IET Collaborative Intelligent Manufacturing*, 2022.

L. Maciaszek, Tomasz Skalniak, and Grzegorz Biziel. "Architectural Principles for Service Cloud Applications." *International Symposium on Business Modeling and Software Design*, 2014.

Liang-Jie Zhang, and Qun Zhou. "CCOA: Cloud Computing Open Architecture." *IEEE International Conference on Web Services*, 2009.

M. Papazoglou. "Cloud Blueprints for Integrating and Managing Cloud Federations." *Software Service and Application Engineering*, 2012.

Mohammad Y. Hajjat, Xin Sun, Y. Sung, D. Maltz, Sanjay G. Rao, K. Sripanidkulchai, and Mohit Tawarmalani. "Cloudward Bound: Planning for Beneficial Migration of Enterprise Applications to the Cloud." *Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, 2010.

Nane Kratzke, and R. Peinl. "ClouNS - a Cloud-Native Application Reference Model for Enterprise Architects." *2016 IEEE 20th International Enterprise Distributed Object Computing Workshop (EDOCW)*, 2016.

Ovidiu Noran, and P. Bernus. "Business Cloudification - An Enterprise Architecture Perspective." *International Conference on Enterprise Information Systems*, 2017.

Srikanth Gurram. "Multi-Cloud Architectures: Principles, Implementation and Strategic Benefits." *International Journal of Advanced Research in Science, Communication and Technology*, 2025.

Thomas Erl, R. Puttini, and Z. Mahmood. "Cloud Computing: Concepts, Technology & Architecture," 2013.

E Report zu multi-cloud Architecture-Principles Literaturrecherche v2

Research-Report generiert mit elicit.ai am 30.12.2025 basierend auf der Frage «What are possible architecture principles for multi-cloud ready enterprise architecture and solution architecture?»

What are possible architecture princples for multi-cloud ready enterprise architecture and solution architecture?

Multi-cloud ready enterprise and solution architectures require five categories of principles: design and abstraction principles (service abstraction, modularization, API integration), scalability and resilience principles (auto-scaling, fault tolerance, high availability), security and governance principles (encryption, policy-based governance, unified identity management), deployment and integration principles (containerization, microservices, infrastructure as code, continuous delivery), and cost and flexibility principles (vendor independence, elasticity, cost optimization), operationalized through technologies including Kubernetes, service mesh, and cloud management platforms.

Abstract

This systematic review of 10 sources identified comprehensive architectural principles for multi-cloud ready enterprise and solution architecture organized across five core themes: design and abstraction principles (service abstraction , architectural abstraction , modularization), scalability and resilience principles (auto-scaling , fault tolerance , high availability), security and governance principles (data encryption , policy-based governance), deployment and integration principles (containerization , microservices architectures , continuous delivery , infrastructure as code), and cost and flexibility principles (vendor independence , elasticity). These principles are operationalized through convergent technologies including containerization, Kubernetes orchestration, service mesh, and cloud management platforms , enabling key multi-cloud capabilities such as portability across platforms (explicitly mentioned in 4 studies), vendor lock-in avoidance (8 studies), and cross-cloud resource orchestration (7 studies).

The evidence base varies substantially in strength, with two empirical studies providing validated migration patterns through case studies and industry projects , while five studies offer theoretical frameworks and four provide literature syntheses . The principles address persistent challenges including management complexity (8 studies), vendor dependency (7 studies), and scalability (7 studies). While fundamental principles around abstraction, modularization, and containerization are well-established, the evidence reveals limited guidance on when to apply specific principles in different contexts, with only the empirical migration studies providing situational pattern selection frameworks. Organizations should prioritize empirically validated patterns for application architecture decisions while leveraging comprehensive frameworks for enterprise-level strategy, recognizing that implementation remains highly context-dependent based on organizational requirements, workload characteristics, and regulatory constraints.

Paper search

We performed a semantic search using the query ”What are possible architecture princples for multi-cloud ready enterprise architecture and solution architecture?” across over 138 million academic papers from the Elicit search engine, which includes all of Semantic Scholar and OpenAlex.

We retrieved the 50 papers most relevant to the query.

Screening

We screened in sources based on their abstracts that met these criteria:

- **Multi-cloud or Hybrid Cloud Focus**: Does the study focus on multi-cloud or hybrid cloud enterprise architecture (rather than solely on single cloud provider architectures)?
- **Architecture Principles Content**: Does the research present, evaluate, or discuss architecture principles, frameworks, or design patterns?

- **Enterprise-level Architecture Scope**: Does the study target enterprise-level or organizational-level architecture (rather than individual application architectures)?
- **Publication Quality**: Is the study a peer-reviewed journal article, conference paper, systematic review, or meta-analysis?
- **Solution Architecture Context**: Does the study address solution architecture in multi-cloud contexts?
- **Architectural vs Technical Implementation Focus**: Does the study focus on architectural principles rather than being limited to technical implementation details, coding practices, or specific cloud service configurations?
- **Architectural vs Operational Focus**: Does the study address architectural principles rather than focusing exclusively on operational concerns (security, compliance, cost optimization) without architectural context?

We considered all screening questions together and made a holistic judgement about whether to screen in each paper.

Data extraction

We asked a large language model to extract each data column below from each paper. We gave the model the extraction instructions shown below for each column.

- **Architecture Principles**:

  Extract all architecture principles, guidelines, or design rules mentioned in the study. Include:

- Exact principle statements or names (quote when possible)
- Core design rules or guidelines proposed
- Architectural directives or recommendations
- Fundamental design concepts advocated
- Any principle-level guidance for multi-cloud architectures Note if principles are explicitly stated vs. implied from the architectural approach described.

- **Multi-Cloud Capabilities**:

  Extract what multi-cloud capabilities these principles enable or support:

- Portability across cloud platforms
- Vendor lock-in avoidance
- Interoperability between clouds
- Cross-cloud resource orchestration
- Multi-cloud deployment flexibility
- Cloud provider abstraction
- Cross-cloud data management
- Multi-cloud resilience or redundancy Specify which capabilities are explicitly mentioned vs. implied.

- **Implementation Approaches**:

  Extract how the principles are operationalized through:

- Architectural patterns (microservices, service-oriented architecture, etc.)
- Technologies or platforms (containerization, APIs, etc.)
- Design methodologies or frameworks
- Specific architectural components or layers
- Integration strategies or mechanisms

- Development and deployment practices Focus on concrete implementation guidance rather than abstract concepts.
- **Challenges Addressed**:

  Extract what specific multi-cloud or enterprise architecture challenges these principles aim to solve:

- Technical challenges (integration, data consistency, security, etc.)
- Business challenges (vendor dependency, cost optimization, etc.)
- Operational challenges (management complexity, monitoring, etc.)
- Architectural challenges (coupling, scalability, maintainability, etc.) Include both explicitly stated problems and those implied by the solutions proposed.
- **Evidence Type**:

  Categorize the type of evidence supporting the principles:

- Theoretical/conceptual framework
- Empirical study with data
- Case study or practical example
- Literature review synthesis
- Industry best practices
- Expert opinion or experience
- Proof of concept or prototype Note the strength and source of evidence for each principle where available.
- **Architecture Scope**:

  Determine the architectural level(s) these principles apply to:

- Enterprise architecture level (business, information, application, technology layers)
- Solution architecture level (specific application or system design)
- Both enterprise and solution levels
- Infrastructure/platform architecture
- Application architecture specifically Extract any guidance about when to apply principles at different architectural levels.

Characteristics of Included Studies

Study	Full text retrieved?	Evidence Type	Architecture Scope	Year
Phani Sekhar Emmanni et al.	No	Theoretical/conceptual framework	Application architecture	2020
Pooyan Jamshidi et al. (2015)	Yes	Empirical study with data, industry best practices, literature review synthesis, expert opinion, case study	Application architecture	2015

Study	Full text retrieved?	Evidence Type	Architecture Scope	Year
Pooyan Jamshidi et al. (2017)	Yes	Empirical study with data, industry best practices, case study, literature review synthesis	Application architecture	2017
Prem Nishanth Kothandarama et al.	No	Literature review synthesis	Both enterprise and solution levels; infrastructure/platform architecture	2025
Karwan Jameel Merseedi et al.	Yes	Literature review synthesis	Both enterprise and solution levels	2024
Srikanth Gurram et al.	No	Theoretical/conceptual framework	Both enterprise and solution levels; application architecture	2025
Omoniyi Babatunde et al.	Yes	Literature review synthesis	Both enterprise and solution levels	2024
C. Pahl et al.	Yes	Theoretical/conceptual framework	Both enterprise and solution levels	2018
Longji Tang et al.	No	Theoretical/conceptual framework	Enterprise architecture level	2010
Ovidiu Noran et al.	No	Theoretical/conceptual framework	Both enterprise and solution levels	2017

The included studies span from 2010 to 2025, with five studies providing full text access. Evidence types are distributed between theoretical/conceptual frameworks (five studies), literature review syntheses (four studies), and empirical studies with multiple evidence sources (two studies). Architecture scope varies, with three studies focusing specifically on application architecture , one on enterprise architecture level , and six addressing both enterprise and solution levels .

Thematic Analysis

Core Architectural Principles

The studies identified a comprehensive set of architectural principles organized around several key themes. **Design and abstraction principles** form the foundation, with multiple studies emphasizing service abstraction , architectural abstraction , and API integration as fundamental to multi-cloud readiness. Modularization and separation of concerns emerged as a consistent theme, with principles advocating for reorganizing multi-tier applications into disjoint groups of service components .

Scalability and resilience principles received extensive attention across studies. Scalability was identified as a critical principle requiring systems to increase resources to accommodate growing workloads , with implementations through load balancing, auto-scaling, and failover mechanisms . Resilience manifested through principles for ensur-

ing high availability , fault tolerance , and distribution of workloads to improve resilience against outages or service interruptions .

Security and governance principles appeared consistently, with multiple studies identifying security as a priority , including principles for data encryption , identity and access management , and unified security guidelines . Policy-based governance and intelligent automation were highlighted as emerging principles for managing multi-cloud complexity.

Deployment and integration principles focused on operational flexibility. Studies advocated for containerization , microservices architectures , and continuous delivery models . Infrastructure as code and service mesh technologies were identified as key enablers of consistency across heterogeneous environments. Integration principles emphasized cloud resource orchestration , data interoperability , and seamless integration of services across different platforms .

Cost and flexibility principles addressed business concerns through principles of cost-effectiveness , vendor independence , and the ability to select services from multiple providers based on specific requirements . Elasticity principles called for adjusting resources automatically in response to changing requirements .

Multi-Cloud Capabilities Enabled

The architectural principles support a comprehensive set of multi-cloud capabilities organized across technical, operational, and business dimensions.

Capability	Studies Explicitly Mentioning	Studies Implying
Portability across cloud platforms	4	3
Vendor lock-in avoidance	3	5
Interoperability between clouds	4	4
Cross-cloud resource orchestration	4	3
Multi-cloud deployment flexibility	5	3
Cloud provider abstraction	2	6
Cross-cloud data management	2	5
Multi-cloud resilience or redundancy	5	3

Portability across cloud platforms was the most explicitly discussed capability, with four studies directly addressing it . Multi-cloud deployment flexibility and resilience/redundancy each received explicit mention in five studies . Cloud provider abstraction was frequently implied across six studies but explicitly discussed in only two , suggesting this capability is often achieved indirectly through other architectural principles rather than being a primary design goal.

The capability for cross-cloud resource orchestration appeared in four studies explicitly , with container orchestration and infrastructure as code identified as key enabling technologies . Interoperability between clouds was supported through service mesh architectures , APIs , and standardized interfaces .

Implementation Approaches

The studies identified concrete implementation approaches spanning architectural patterns, technologies, design methodologies, and integration strategies.

Architectural patterns centered on microservices and service-oriented architecture as foundational patterns. Service-based cloud architecture migration patterns were detailed as fine-grained, reusable, and composable architectural change patterns . Refactoring into fine-grained components and reorganizing applications into disjoint service groups emerged as common pattern-based approaches.

Technologies and platforms showed strong convergence around containerization as a core enabling technology. Kubernetes appeared in multiple contexts for container orchestration , with Platform-as-a-Service (PaaS) clouds like Microsoft Azure specifically mentioned . Cloud management platforms (CMPs) were identified as essential orchestration tools, alongside infrastructure as code and service mesh technologies . APIs and platform-agnostic interfaces enable abstraction and interoperability.

Design methodologies and frameworks varied in formality. Holistic frameworks for cloud-native development and multidimensional frameworks covering architectural abstraction, developer experience, data management, performance engineering, and security integration represented comprehensive approaches. Assembly-based situational migration at the architecture level and variability models for pattern selection provided structured methodologies for migration planning.

Integration strategies emphasized intelligent automation and policy-based governance for managing multi-cloud complexity. Resilient orchestration and management solutions , unified identification and access control , and integration frameworks combining cloud management platforms, unified identity frameworks, data orchestration tools, and software-defined networking formed the operational foundation. Continuous integration/continuous delivery (CI/CD) automation streamlined deployment practices.

Specific architectural components included load balancing, auto-scaling, and failover mechanisms for scalability . Data synchronization tools and migration transition graphs supported cross-cloud data management and migration planning. Emerging approaches highlighted AI-driven orchestration, decentralized control planes, and edge native deployments as future directions.

Challenges Addressed

The architectural principles directly respond to a comprehensive set of challenges organized across technical, business, operational, and architectural dimensions.

Technical challenges dominated the discussion, with integration challenges appearing in seven studies . Security concerns were identified in six studies , encompassing data encryption, identity and access management, and unified security policies. Data consistency challenges appeared in five studies . Technical compatibility , interoperability standards , stateful application migration , and data gravity represented more specialized technical challenges.

Business challenges focused primarily on vendor dependency, explicitly mentioned in seven studies . Cost optimization appeared in six studies , reflecting the economic imperative driving multi-cloud adoption. Regulatory compliance and regulatory agility emerged as business challenges requiring architectural solutions.

Operational challenges centered on management complexity, identified in eight studies . Monitoring challenges appeared in five studies . Administrative burden of managing multiple cloud relationships , workflow scheduling , and governance inconsistencies represented operational complexity at different organizational levels.

Architectural challenges emphasized scalability in seven studies and maintainability in six studies . Coupling challenges appeared in five studies , reflecting concerns about architectural rigidity. High availability , elasticity , and fault tolerance represented quality attributes requiring architectural support.

Synthesis

While the studies show substantial agreement on core principles, differences emerge in emphasis and architectural scope that reflect the evolution of multi-cloud thinking from 2010 to 2025.

Scope evolution : Earlier work like Longji Tang et al. (2010) focused on enterprise architecture levels, extending service-oriented architecture to cloud contexts at a conceptual level. The 2015-2017 Jamshidi studies emphasized application architecture specifically, providing concrete migration patterns with empirical validation. Recent reviews (2024-2025) address both enterprise and solution levels, reflecting maturation toward holistic approaches that bridge strategic and tactical concerns.

Evidence foundations : The two empirical studies by Jamshidi et al. provide the strongest evidence base, combining empirical data from migration projects, case studies, industry best practices, and literature synthesis. These studies identified specific patterns like reorganizing multi-tier applications into disjoint service components and incremental migration approaches . In contrast, theoretical frameworks and literature reviews synthesize broader principles but with less validation of their effectiveness in practice.

Technology emphasis : Earlier conceptual work emphasized service-oriented architecture principles without specific implementation guidance. Studies from 2015-2018 introduced containerization and microservices as key patterns. Recent literature reviews (2024-2025) converge on a standard technology stack: containerization, Kubernetes for orchestration, infrastructure as code, service mesh, and cloud management platforms . This convergence suggests these technologies have become established best practices rather than experimental approaches.

Capability priorities : All studies addressing capabilities explicitly mention portability and multi-cloud deployment flexibility as primary benefits. However, cloud provider abstraction, despite being fundamental to multi-cloud readiness, was explicitly discussed in only two studies while implied in six others . This suggests abstraction is often achieved indirectly through containerization and standardized interfaces rather than being explicitly designed for.

Challenge context : The challenges addressed show remarkable consistency across studies, with management complexity appearing in eight studies , vendor dependency in seven , and scalability in seven . This consistency validates that these are persistent challenges requiring architectural solutions. However, newer studies introduce emerging challenges like data gravity , stateful application migration , and the need for AI-driven orchestration , reflecting increasing architectural sophistication.

Principle application guidance : Studies differ in specificity of when to apply principles. The Jamshidi empirical work provides situational guidance through pattern selection frameworks and variability models , allowing organizations to select patterns based on specific contexts. Theoretical frameworks present principles more generally without contextual guidance. Recent reviews acknowledge this gap, discussing the need for policy-based governance and intelligent automation to guide principle application, but without providing specific decision criteria.

The synthesis reveals that while fundamental principles around abstraction, modularization, and containerization are well-established and empirically validated, implementation remains context-dependent. Organizations adopting multi-cloud architectures should prioritize empirically validated patterns from migration studies for application architecture decisions, while leveraging comprehensive frameworks from recent reviews for enterprise-level strategy. The convergence on containerization, Kubernetes, and infrastructure as code as enabling technologies suggests these should form the technological foundation, with specific pattern selection guided by organizational context, workload characteristics, and regulatory requirements.

References

C. Pahl, Pooyan Jamshidi, and O. Zimmermann. “Architectural Principles for Cloud Software.” *ACM Trans. Internet Techn.*, 2018.

Karwan Jameel Merseedi, and Dr. Subhi R. M. Zeebaree. “Cloud Architectures for Distributed Multi-Cloud Computing: A Review of Hybrid and Federated Cloud Environment.” *Indonesian Journal of Computer Science*, 2024.

Longji Tang, Jing Dong, Yajing Zhao, and Liang-Jie Zhang. “Enterprise Cloud Service Architecture.” *IEEE International Conference on Cloud Computing*, 2010.

Omoniyi Babatunde, Jeremiah Olamijuwon, Emmanuel Cadet, Olajide Soji Osundare, and Zein Samira. “Designing Multi-Cloud Architecture Models for Enterprise Scalability and Cost Reduction.” *Open Access Research Journal of Engineering and Technology*, 2024.

Ovidiu Noran, and P. Bernus. “Business Cloudification - An Enterprise Architecture Perspective.” *International Conference on Enterprise Information Systems*, 2017.

Phani Sekhar Emmanni. “Architecting Cloud-Native Applications for Multi-Cloud Environments.” *International Journal of Science and Research (IJSR)*, 2020.

Pooyan Jamshidi, C. Pahl, and N. Mendonça. “Pattern-based Multi-cloud Architecture Migration.” *Software, Practice & Experience*, 2017.

Pooyan Jamshidi, C. Pahl, Samuel J. Chinenyeze, and Xiaodong Liu. “Cloud Migration Patterns: A Multi-Cloud Service Architecture Perspective.” *ICSOC Workshops*, 2015.

Prem Nishanth Kothandarama. “Designing Developer Platforms for Cross-Cloud Portability and Scale.” *International Research Journal on Advanced Engineering Hub (IRJAEH)*, 2025.

Srikanth Gurram. “Multi-Cloud Architectures: Principles, Implementation and Strategic Benefits.” *International Journal of Advanced Research in Science, Communication and Technology*, 2025.

F Report zu Quality Attributes Recherche

Research-Report generiert mit elicit.ai am 01.01.2026 basierend auf der Frage «Which architecture-related quality attributes are suitable for the comparative evaluation of enterprise architectures in technology-agnostic multi-cloud environments? »

Which architecture-related quality attributes are suitable for the comparative evaluation of enterprise architectures in technology-agnostic multi-cloud environments?

Performance, cost-effectiveness, reliability, security, scalability, and availability constitute the core quality attributes suitable for technology-agnostic multi-cloud evaluation, with interoperability and portability serving as essential additional attributes for addressing multi-cloud-specific challenges of provider heterogeneity and cross-environment migration.

Abstract

This systematic review of 25 sources identifies a core set of architecture-related quality attributes suitable for comparative evaluation in technology-agnostic multi-cloud environments. Performance and cost-effectiveness emerge as the most consistently cited attributes, each appearing in 11 sources , followed by reliability and security (7 sources each) and scalability (6 sources) . Empirical evidence from a survey of 42 automotive practitioners quantifies the relative importance of these attributes, with availability rated highest (x=6.7), followed by reliability (x=6.5), security (x=6.3), and scalability (x=6.0) . For multi-cloud-specific evaluation, interoperability and portability address the unique challenges of heterogeneity and cross-provider migration.

The evidence supports the suitability of these attributes for technology-agnostic comparison through validated evaluation methods including cloud-independent resource classification schemes , cloud-agnostic evaluation matrices , and fuzzy-based multi-criteria decision making approaches that accommodate uncertainty in cross-provider assessments . Quantitative validation demonstrates practical applicability, with implementations achieving 40% reduction in deployment inconsistencies, 60% faster incident resolution, and 25% improvement in cost optimization across multiple industries . However, attribute prioritization is context-dependent: retail organizations benefit from modular architectures emphasizing flexibility, while financial services require integral architectures emphasizing consistent integration , indicating that while the core quality attributes are broadly applicable, their relative weighting must be calibrated to specific organizational and regulatory contexts.

Paper search

We performed a semantic search using the query ”Which architecture-related quality attributes are suitable for the comparative evaluation of enterprise architectures in technology-agnostic multi-cloud environments?” across over 138 million academic papers from the Elicit search engine, which includes all of Semantic Scholar and OpenAlex.

We retrieved the 499 papers most relevant to the query.

Screening

We screened in sources based on their abstracts that met these criteria:

- **Enterprise Architecture Evaluation Focus**: Does the study focus on enterprise architecture evaluation or assessment and address architecture-related quality attributes (e.g., scalability, reliability, security, interoperability, maintainability)?
- **Multi-Cloud Context**: Does the study involve multi-cloud environments, hybrid cloud architectures, or cloud-agnostic approaches (rather than focusing solely on single-cloud or on-premises architectures)?

- **Comparative Evaluation Methods**: Does the research address comparative evaluation methods, frameworks, or metrics for architectural assessment?
- **Empirical Evidence or Systematic Analysis**: Is the study an empirical study (case study, experiment, survey) or a systematic review/meta-analysis on enterprise architecture evaluation (rather than an opinion paper, editorial, or purely theoretical discussion without empirical evidence)?
- **Technology-Agnostic Approach**: Does the study address technology-agnostic architectural approaches or platform-independent design principles (rather than being limited to specific technology implementations without architectural abstraction)?
- **Enterprise Architecture Level**: Does the study focus on enterprise architecture rather than software architecture at the application level only?

We considered all screening questions together and made a holistic judgement about whether to screen in each paper.

Data extraction

We asked a large language model to extract each data column below from each paper. We gave the model the extraction instructions shown below for each column.

- **Quality Attributes**:

  Extract all architecture-related quality attributes mentioned in the study including:

- Specific quality attribute names (e.g., availability, reliability, security, performance, scalability, cost-effectiveness)
- How each quality attribute is defined or characterized
- Whether the attribute is explicitly stated as suitable for enterprise architecture comparison
- Any categorization or grouping of quality attributes provided
- Quality metrics or measurement approaches associated with each attribute

- **Evaluation Methods**:

  Extract details about methods used to assess or compare quality attributes including:

- Specific evaluation frameworks, methodologies, or approaches used
- Assessment criteria and decision-making processes
- Comparison techniques (e.g., multi-criteria evaluation, fuzzy logic, benchmarking)
- Tools or instruments used for evaluation
- Validation methods or empirical evidence provided
- Any limitations or challenges in the evaluation approach mentioned

- **Multi-Cloud Context**:

  Extract information about the multi-cloud environment and context including:

- Type of multi-cloud deployment or scenario described
- Cloud providers or platforms involved
- Integration challenges or complexity factors mentioned
- Vendor lock-in considerations or technology independence requirements
- Heterogeneity aspects (technology stacks, services, pricing models)
- Migration or portability considerations between cloud environments

- **Architecture Patterns**:

  Extract details about enterprise architecture types and patterns including:

- Specific architecture styles or patterns mentioned (e.g., SOA, microservices, event-driven)
- Architectural components or layers described
- Technology-agnostic design approaches or principles
- Legacy system integration patterns
- Infrastructure-as-Code or deployment patterns
- Any architectural frameworks or reference models used

- **Suitability Evidence**:

  Extract evidence supporting the suitability of quality attributes for comparative evaluation including:

- Empirical results or performance improvements achieved
- Quantitative metrics or measurements provided
- Success factors or benefits demonstrated
- Industry validation or case study evidence
- Comparative advantages over alternative approaches
- Practical applicability across different enterprise contexts
- Any identified gaps or areas needing further research

- **Study Context**:

  Extract contextual information about the research including:

- Research methodology (survey, case study, framework development, etc.)
- Industry domains or application areas studied
- Sample size and participant characteristics (if applicable)
- Geographic scope or organizational types involved
- Time period of study or implementation
- Technology maturity level or adoption stage addressed

Characteristics of Included Studies

The following table presents the characteristics of all 25 sources included in this review, examining their research focus, methodology, and relevance to quality attributes in multi-cloud enterprise architecture evaluation.

Study	Full text retrieved?	Study Type	Research Methodology	Industry Domain	Multi-Cloud Focus
Papaioannou et al., 2013	Yes	Primary study	Framework development, case study	Cloud computing, distributed applications	Yes - multi-cloud deployment evaluation

Study	Full text retrieved?	Study Type	Research Methodology	Industry Domain	Multi-Cloud Focus
Bhargav Sai et al., 2025	No	Primary study	Framework development	Fintech, pharmaceutical, healthcare, retail, manufacturing	Yes - multi-cloud integration
Banijamali et al., 2019	Yes	Primary study	Online survey	Automotive cloud platforms	Limited
Menzel et al., 2011	No	Primary study	Framework development	Enterprise application migration	Yes - multi-component migration
Hussain et al., 2025	No	Primary study	Mixed-methods research	Financial services, healthcare, manufacturing, public sectors	Limited
Hannemann et al., 2022	No	Primary study	Decision model development, literature review	Enterprise architecture, digital transformation	Limited
Ramchand et al., 2017	No	Primary study	Framework development	Cloud migration	Yes - public/private/hybrid cloud
Aruna et al., 2016	No	Primary study	Framework development	Cloud computing, IaaS	Yes - federated cloud architecture
Cao et al., 2022	No	Primary study	Survey	Retail, financial services	Yes - hybrid multi-cloud
Ghazaryan et al., 2025	No	Review	Literature review, meta-analysis	Cloud computing	Yes - multi-cloud deployment models
Papaioannou et al., 2013a	No	Primary study	Framework development	Cloud computing, distributed applications	Yes - cloud federations
Mirsalari et al., 2020	Yes	Primary study	Mixed method (SLR + survey)	Enterprise architecture	Limited
Ardagna et al., 2012	Yes	Primary study	Framework development	Cloud computing, software development	Yes - multiple clouds
Sodhi et al., 2012	No	Primary study	Framework development	Enterprise computing platforms	Partial - virtualization and cloud

Study	Full text retrieved?	Study Type	Research Methodology	Industry Domain	Multi-Cloud Focus
Assis et al., 2016	No	Review	Survey	Cloud federations	Yes - cloud federations
Chauhan et al., 2014	No	Review	Systematic mapping study	Multiple domains	Limited
Razavi et al., 2009	No	Primary study	Framework development, case study	Enterprise architecture	No
Evangelinou et al., 2015	No	Primary study	Framework development	Cloud computing, legacy migration	Yes - multi-cloud deployment
Lima et al., 2019	No	Primary study	Mapping study, survey	Software ecosystems	Limited
Bastani et al., 2011	No	Primary study	Framework development	Enterprise service-oriented computing	Limited
Serhiienko et al., 2018	No	Primary study	Framework development, experimentation	Multi-cloud management	Yes - hybrid cloud, multi-cloud
Zimmermann et al., 2011	No	Primary study	Framework development	Service-oriented architectures	Limited
Davoudi et al., 2012	No	Primary study	Framework development	Enterprise architecture	No
Masuda et al., 2017	No	Primary study	Survey, case study, framework development	Pharmaceutical	Partial - cloud/mobile IT
Joukov et al., 2013	No	Primary study	Case study	Enterprise IT, cloud interoperability	Yes - multi-cloud interoperability

The included studies span from 2009 to 2025, with a concentration in the 2011-2019 period reflecting the maturation of cloud computing research. Of the 25 sources, only 4 provided full text access. The predominant research methodology across studies is framework development, with 15 studies employing this approach. Fourteen studies demonstrate explicit multi-cloud focus, while the remainder address broader enterprise architecture or single-cloud contexts with transferable findings.

Quality Attributes Identified

Overview of Quality Attributes

The analysis identified a diverse set of quality attributes across the literature. The following table consolidates all quality attributes explicitly mentioned in the sources, organized by frequency of citation.

Quality Attribute	Sources Citing	Specific Context/Definition
Performance	Papaioannou 2013 , Bhargav Sai 2025 , Banijamali 2019 , Hussain 2025 , Ramchand 2017 , Ghazaryan 2025 , Papaioannou 2013a , Ardagna 2012 , Evangelinou 2015 , Bastani 2011 , Serhiienko 2018	Throughput, response time ; network performance ; ability to adapt to workloads
Reliability	Papaioannou 2013 , Bhargav Sai 2025 , Banijamali 2019 , Ramchand 2017 , Papaioannou 2013a , Mirsalari 2020 , Chauhan 2014	Meeting service-level objectives ; importance rating x=6.5 in automotive context
Security	Bhargav Sai 2025 , Banijamali 2019 , Hussain 2025 , Ghazaryan 2025 , Mirsalari 2020 , Ardagna 2012 , Bastani 2011	Ensured by Infrastructure-as-Code standards ; importance rating x=6.3
Scalability	Bhargav Sai 2025 , Banijamali 2019 , Hussain 2025 , Mirsalari 2020 , Ardagna 2012 , Bastani 2011	Elastic scalability ; challenges with dynamic vehicle numbers
Cost/Cost-effectiveness	Papaioannou 2013 , Bhargav Sai 2025 , Menzel 2011 , Ramchand 2017 , Aruna 2016 , Ghazaryan 2025 , Papaioannou 2013a , Ardagna 2012 , Evangelinou 2015 , Serhiienko 2018 , Joukov 2013	Comparing deployment options ; cost metrics for provider comparison
Availability	Banijamali 2019 , Ardagna 2012	Most important QA in automotive domain (x=6.7)
Interoperability	Banijamali 2019 , Joukov 2013	Importance rating x=6.0 ; implied through non-interoperability challenges
Maintainability/Portability	Mirsalari 2020	Combined attribute in EA evaluation model
Reusability	Mirsalari 2020	Part of seven main EA quality attributes
Service Adaptability	Chauhan 2014	Commonly addressed in cloud-based systems
Energy Optimization	Chauhan 2014	Emerging quality attribute in cloud systems
Regulatory Compliance	Ghazaryan 2025	Deployment model selection factor
Consistency Guarantees	Evangelinou 2015	Cloud service differentiation factor
Fault Tolerance	Evangelinou 2015	Key QoS consideration
Architectural Sustainability	Hussain 2025	Long-term value consideration
Business Value Alignment	Hussain 2025	Strategy effectiveness factor

Performance emerges as the most frequently cited quality attribute, appearing in 11 of 25 sources. Cost-effectiveness follows closely, cited in 11 sources, reflecting the practical importance of financial considerations in cloud deployment

decisions. Security and reliability each appear in 7 sources, while scalability is mentioned in 6 sources.

Quality Attribute Rankings and Importance

Empirical evidence from the automotive cloud platform survey provides quantitative importance ratings for quality attributes using a weighted arithmetic mean: availability scored highest at x=6.7, followed by reliability at x=6.5, security at x=6.3, and both scalability and interoperability at x=6.0 . These ratings were derived from 42 valid survey responses from practitioners and researchers with practical experience in architecture design .

The enterprise architecture quality model developed through systematic literature review and factor analysis identifies seven main quality attribute categories encompassing 30 specific indicators: alignment and integrity, quality of EA products and services, security, maintainability and portability, reliability, reusability, and scalability . This categorization addresses multiple aspects of enterprise architecture evaluation, though it was not specifically validated for multi-cloud contexts .

Technology-Agnostic Considerations

Several sources emphasize the importance of technology-agnostic approaches for quality attribute evaluation. Cloud-agnostic evaluation matrices enable comparison of provider capabilities across workload requirements, service compatibility, compliance standards, network performance, and cost metrics . A cloud-independent resource classification scheme facilitates reasoning about multi-cloud deployments of complex large-scale applications . The MODA-Clouds framework explicitly supports cloud-agnostic design through model-driven development, allowing systems to be designed independently of specific cloud technologies .

Evaluation Methods for Quality Attributes

Multi-Criteria Decision Making Approaches

Method	Sources	Application Context	Key Characteristics
Analytic Hierarchy Process (AHP)	Menzel 2011 , Razavi 2009 , Davoudi 2012	Cloud service selection , EA scenario evaluation	Multi-criteria-based selection algorithm
Fuzzy AHP	Razavi 2009 , Davoudi 2012 , Sodhi 2012	EA analysis under uncertainty , platform suitability assessment	Addresses vagueness in expert judgments
Fuzzy Sets Ranking	Aruna 2016	Service provider selection in federated clouds	Problem decomposition, priority judgment, aggregation
DEMATEL + PROMETHEE	Hannemann 2022	DT technology influence on EA principles	Multi-criteria decision-making

Fuzzy-based approaches appear prominently in the literature for handling uncertainty in quality attribute assessment. Fuzzy logic addresses the inherent vagueness in expert judgments when comparing enterprise architecture scenarios . The fuzzy sets technique produces ordered rankings of platforms based on quality attribute criteria, demonstrating efficacy across virtualization and cloud platforms .

Benchmarking and Empirical Evaluation

The SPEC jEnterprise2010 distributed benchmark serves as a case study for evaluating deployment options based on performance, reliability, and cost . This benchmark-based approach enables quantitative comparison of multi-cloud deployments through standardized performance metrics such as EjOPS (enterprise Java operations per second) and cost per hour .

A joint benchmarking and optimization methodology supports design-time assessment of multi-cloud applications by identifying deployments with minimum costs while maintaining QoS guarantees . Cloud benchmarks help developers choose appropriate architectures and services by evaluating cost, performance, consistency guarantees, load-balancing, caching, fault tolerance, and SLAs .

Architecture evaluation methods in the automotive domain include active reviews from intermediate design (ARID) and scenario-based architecture analysis method (SAAM), with testing approaches such as automated test-benches, stress tests, and hardware-in-the-loop testing . Challenges identified include availability and quality of reviewers, adaptation to agile methodologies, and difficulties comparing design alternatives .

Framework-Based Evaluation

Framework	Purpose	Key Components
CloudGenius	Cloud migration decision support	AHP-based selection, CumulusGenius prototype
MODAClouds	Multi-cloud design and execution	Decision Support System, CORAS, PREDIQT, MDQP techniques
ESARC	Service-oriented architecture assessment	Standardized classification scheme
Adaptive Integrated EA	Cloud/mobile IT strategy support	Comparison with TOGAF characteristics

The MODAClouds framework incorporates established model-driven risk and quality analysis methods including CORAS and PREDIQT, along with Model Driven Quality Prediction techniques such as PUMA, PCM, and KlaperSuite . The Decision Support System enables risk analysis for cloud provider selection and evaluation of cloud adoption impact on business processes .

Multi-Cloud Context and Heterogeneity

Types of Multi-Cloud Deployments

The literature addresses several multi-cloud deployment scenarios:

- **Federated cloud architecture** : A heterogeneous and distributed model aggregating different IaaS providers
- **Cloud federations** : Including horizontal federation, InterClouds federation, Cross-Clouds, and Sky computing
- **Hybrid multi-cloud environments** : Combining different cloud services for complex interdependent systems
- **Public, private, and hybrid cloud selection** : Decision frameworks for choosing appropriate deployment models

Integration Challenges and Complexity

Key challenges in multi-cloud environments include inconsistent governance models, vendor lock-in risks, and integration challenges that hinder operational efficiency . Enterprise software and middleware non-interoperability creates complexity beyond cloud provider-specific issues . The heterogeneity of cloud resources presents challenges for application developers choosing appropriate providers and resources .

Cloud-independent resource classification addresses heterogeneity by classifying VM types across providers based on performance metrics including CPU, memory, and I/O capability . Abstract APIs for deployment and management support technology independence and facilitate migration between cloud environments .

Vendor Lock-in Considerations

Technology independence requirements are addressed through cloud-agnostic evaluation matrices , cloud-independent classification schemes , and abstract APIs for cross-cloud deployment . Abstraction libraries aim to address vendor lock-in in multi-cloud management platforms .

Architecture Patterns for Multi-Cloud Environments

Identified Architecture Styles

Architecture Style	Sources	Application Context
Service-Oriented Architecture (SOA)	Banijamali 2019 , Zimmermann 2011 , Bastani 2011	Automotive cloud platforms , enterprise systems
Event-Driven Architecture	Banijamali 2019 , Bhargav Sai 2025	QA fulfillment , legacy integration
Microservices Architecture	Banijamali 2019 , Cao 2022	Modular design, cloud platforms
Enterprise Service-Oriented Architecture (ESOA)	Bastani 2011	Enterprise architectural style
Enterprise Cloud Service Architecture (ECSA)	Bastani 2011	Hybrid cloud-ESOA style
Model-Driven Development (MDD)	Ardagna 2012	Cloud-agnostic design

Event-driven and service-oriented architecture are the most applied design styles for fulfilling quality attributes in automotive cloud platforms . Additional styles identified include multi-layered, client-server, pipeline, Lambda, and message bus architectures .

Technology-Agnostic Design Principles

Key technology-agnostic design approaches include:

- Cloud-independent resource classification enabling reasoning across providers
- Infrastructure-as-Code standards ensuring consistency across cloud environments
- Federated integration architectures using event-driven patterns for legacy system integration
- Model-driven development supporting cloud-agnostic design with semi-automatic code translation
- Abstract APIs for deployment and management across multiple clouds
- Strict application documentation and standardization for enterprise applications

Data Architecture Considerations

Enterprise data architecture emerges as crucial digital infrastructure for managing complex multi-cloud systems . Two contrasting frameworks are identified: microservices architecture implementing modular design and single fabric architecture implementing integral design . The choice between modular and integral approaches depends on industry context, with retail establishments benefiting from non-integral (modular) architecture and financial services establishments benefiting from non-modular (integral) architecture for cloud adoption .

Suitability Evidence for Quality Attributes

Quantitative Performance Evidence

Source	Metric	Result
Bhargav Sai 2025	Deployment inconsistency reduction	40% reduction
Bhargav Sai 2025	Incident resolution improvement	60% faster
Bhargav Sai 2025	Cost optimization	25% improvement
Papaioannou 2013	Cost savings from optimal deployment	$2,100 per year
Joukov 2013	Migration cost reduction	Order of magnitude reduction
Aruna 2016	Service provider selection	Improved selection rate, response time, reduced overhead

The decision-making frameworks demonstrated practical benefits across fintech, pharmaceutical, healthcare, retail, and manufacturing organizations . Implementation resulted in operational excellence and business agility through coordinated ecosystem orchestration rather than isolated technology selection .

Industry Validation

Multiple sources provide industry validation through various mechanisms:

- Case studies in global pharmaceutical companies validating adaptive EA frameworks
- Assessment workshops with global vendors of service-oriented platforms validating ESARC
- Analysis of over 210,000 retail and financial services establishments validating data architecture approaches
- Evaluation using three real-life applications from large corporations demonstrating enterprise IT transformation policies
- Use of SPEC jEnterprise2010 benchmark providing standardized industry validation

Cross-Context Applicability

Quality attributes demonstrate applicability across diverse contexts. Performance analysis shows variations in cost efficiency, scalability, and security outcomes across migration strategies, with refactoring and rearchitecting delivering superior long-term value despite higher initial investment . The comprehensive decision framework integrating technical and business dimensions shows strong predictive validity for migration success across financial services, healthcare, manufacturing, and public sectors .

Different data architecture designs complement cloud adoption under different contingencies: retail organizations benefit from decentralized approaches ensuring flexible adaptability to external markets, while financial services benefit from centralized approaches ensuring consistent integration and robust control .

Synthesis

Reconciling Quality Attribute Prioritization

The evidence reveals apparent heterogeneity in quality attribute prioritization across studies. Performance and cost-effectiveness emerge as universally recognized attributes, each cited in 11 sources. However, the relative importance of specific attributes varies by context.

This variation can be explained through **context and population distinctions** . In the automotive domain, availability achieves the highest importance rating (x=6.7) , reflecting the safety-critical nature of vehicle systems. In enterprise IT transformation contexts, cost-effectiveness dominates due to migration budget constraints, with strict standardization achieving order-of-magnitude cost reductions . For digital transformation initiatives, security and compliance become paramount due to regulatory requirements .

The **study quality hierarchy** suggests weighting empirical studies more heavily. The survey of 42 automotive practitioners and the analysis of over 210,000 establishments provide stronger evidence than conceptual frameworks. These empirical studies consistently identify performance, reliability, security, scalability, and cost as core quality attributes.

Technology-Agnostic Attribute Categories

Based on the synthesis of evidence, quality attributes suitable for technology-agnostic multi-cloud evaluation can be organized into four tiers:

Tier 1 - Universally Applicable: Performance , cost-effectiveness , and reliability demonstrate consistent relevance across all multi-cloud contexts regardless of industry domain or deployment model.

Tier 2 - Context-Dependent Critical: Security and scalability are critical in regulated industries and high-growth scenarios respectively. Availability becomes paramount in safety-critical or high-availability requirements.

Tier 3 - Multi-Cloud Specific: Interoperability , portability , and vendor independence address multi-cloud-specific challenges of heterogeneity and migration between providers.

Tier 4 - Emerging Considerations: Energy optimization , service adaptability , and architectural sustainability represent emerging quality attributes gaining importance in modern enterprise architectures.

Evaluation Method Suitability

For technology-agnostic multi-cloud environments, fuzzy-based multi-criteria decision making approaches demonstrate particular suitability by accommodating uncertainty inherent in cross-provider comparisons . The combination of benchmarking for quantitative assessment with structured decision frameworks provides comprehensive evaluation capability.

Cloud-independent classification schemes and abstract APIs enable meaningful comparison across heterogeneous cloud environments by establishing common reference points for quality attribute measurement. The PACE layering

model provides systematic prioritization of modernization efforts , while Infrastructure-as-Code standards ensure consistency in evaluation across diverse cloud platforms .

Practical Implications

The evidence supports a hierarchical approach to quality attribute selection based on organizational context. Organizations should prioritize Tier 1 attributes as baseline requirements, then incorporate Tier 2 attributes based on regulatory and scalability requirements, add Tier 3 attributes for active multi-cloud strategies involving provider migration or hybrid deployments, and consider Tier 4 attributes for forward-looking architectural decisions.

The non-linear relationship between application technical debt and optimal strategy selection suggests that quality attribute priorities should evolve as organizations progress through cloud maturity stages. Organizations at early adoption stages may focus primarily on cost-effectiveness and basic performance metrics, while mature cloud adopters increasingly prioritize architectural sustainability and interoperability .

References

A. Zimmermann, G. Zimmermann, and Zimmermann Und Partner. "ESARC - Enterprise Services Architecture Reference Cube for Capability Assessments of Service-Oriented Systems," 2011.

Ahmad Banijamali, Philipp Heisig, Johannes Kristan, P. Kuvaja, and M. Oivo. "Software Architecture Design of Cloud Platforms in Automotive Domain: An Online Survey." *IEEE International Conference on Service-Oriented Computing and Applications*, 2019.

Antonis Papaioannou. "Design and Implementation of a System for Evaluating Distributed Application Deployments in Multi-Clouds," 2013.

Antonis Papaioannou, and K. Magoutis. "An Architecture for Evaluating Distributed Application Deployments in Multi-Clouds." *IEEE International Conference on Cloud Computing Technology and Science*, 2013.

Athanasia Evangelinou, M. Ciavotta, George Kousiouris, and D. Ardagna. "A Joint Benchmark-Analytic Approach For Design-Time Assessment of Multi-Cloud Applications." *International Conference on Cloud Forward*, 2015.

B. Sodhi, and T. Prabhakar. "Assessing Platform Suitability for Achieving Quality in Guest Applications." *Asia-Pacific Software Engineering Conference*, 2012.

Bhargav Sai, and Pillati. "Leading Through Complexity: Decision-Making Frameworks for Multi-Cloud Integration and Platform Scalability." *Journal of Information Systems Engineering & Management*, 2025.

D. Ardagna, E. D. Nitto, G. Casale, D. Petcu, Parastoo Mohagheghi, Sébastien Mosser, Peter Matthews, et al. "MODACLOUDS: A Model-Driven Approach for the Design and Execution of Applications on Multiple Clouds." *2012 4th International Workshop on Modeling in Software Engineering (MISE)*, 2012.

Erik Ghazaryan. "Comparative Analysis of Cloud Deployment Models: Public, Private, Hybrid, and Multi-Cloud." *International Journal Of Engineering And Computer Science*, 2025.

F. Bastani, and Longji Tang. "Modeling and Analyzing Service-Oriented Enterprise Architectural Styles," 2011.

Izabelle Hannemann, Sarah Rodrigues, E. Loures, F. Deschamps, and J. Cestari. "Applying a Decision Model Based on Multiple Criteria Decision Making Methods to Evaluate the Influence of Digital Transformation Technologies on Enterprise Architecture Principles." *IET Collaborative Intelligent Manufacturing*, 2022.

K. Ramchand, Mohan Baruwal Chhetri, and R. Kowalczyk. "Towards a Flexible Cloud Architectural Decision Framework for Diverse Application Architectures." *ACIS*, 2017.

L. Aruna, and M. Aramudhan. "Framework for Ranking Service Providers of Federated Cloud Architecture Using Fuzzy Sets," 2016.

M. Davoudi, and K. Sheikhvand. "An Approach Towards Enterprise Architecture Analysis Using AHP and Fuzzy

AHP," 2012.
M. M. Assis, and L. Bittencourt. "A Survey on Cloud Federation Architectures: Identifying Functional and Non-Functional Properties." *Journal of Network and Computer Applications*, 2016.
M. Razavi, Shams Fereidoon, Aliee, Amir Esmaeil, and Sarabadani Tafreshi. "A Fuzzy AHP Based Approach Towards Enterprise Architecture Evaluation," 2009.
Michael Menzel, and R. Ranjan. "CloudGenius: Automated Decision Support for Migrating Multi-Component Enterprise Applications to Clouds." *arXiv.org*, 2011.
Mohammad Asad Hussain. "A Comparative Analysis of Cloud Migration Strategies for Enterprise Systems Architecture." *World Journal of Advanced Engineering Technology and Sciences*, 2025.
Muhammad Aufeef Chauhan, and M. Babar. "A Systematic Mapping Study of Software Architectures for Cloud Based Systems," 2014.
N. Joukov, and V. Shorokhov. "Cloud Interoperability via Quick Enterprise Applications Re-Builds." *International Conference on Cloud Computing and Services Science*, 2013.
Oleksii Serhiienko, and Josef Spillner. "Systematic and Recomputable Comparison of Multi-Cloud Management Platforms." *2018 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, 2018.
Ruiqing Cao, and M. Iansiti. "Cloud Adoption and Digital Transformation: The Paradoxical Role of Enterprise Data Architecture." *Social Science Research Network*, 2022.
Seyedeh Reyhaneh Mirsalari, and M. Ranjbarfard. "A Model for Evaluation of Enterprise Architecture Quality." *Evaluation and Program Planning*, 2020.
T. Lima, C. Werner, and R. Santos. "Identifying Architecture Attributes in the Context of Software Ecosystems Based on a Mapping Study." *International Conference on Software Business*, 2019.
Yoshimasa Masuda, Seiko Shirasaka, Shuichiro Yamamoto, and T. Hardjono. "An Adaptive Enterprise Architecture Framework and Implementation: Towards Global Enterprises in the Era of Cloud/Mobile IT/Digital IT." *International Journal of Enterprise Information Systems*, 2017.

G Report zu Trade-Off Kriterien Recherche

Research-Report generiert mit elicit.ai am 02.01.2026 basierend auf der Frage «Which architecture-level trade-offs shape the design and evaluation of enterprise architectures in technology-agnostic multi-cloud environments? »

Which architecture-level trade-offs shape the design and evaluation of enterprise architectures in technology-agnostic multi-cloud environments?

Seven primary architecture-level trade-offs shape multi-cloud enterprise architecture design and evaluation: performance versus portability, cost versus capability (the most frequently addressed), flexibility versus complexity, security versus usability, vendor lock-in versus scalability, modularity versus integration, and consistency versus complexity—with their relative importance and optimal resolution varying by organizational context, industry sector, and workload characteristics.

Abstract

This systematic review of 25 sources identifies seven primary architecture-level trade-offs that shape enterprise architecture design and evaluation in technology-agnostic multi-cloud environments: performance versus portability , cost versus capability , flexibility versus complexity , security versus usability , vendor lock-in versus scalability , modularity versus integration , and consistency versus complexity . The cost versus capability trade-off emerges as most frequently discussed, with quantitative evidence demonstrating that systematic optimization approaches can achieve cost reductions of 39-78% compared to provider best practices and annual savings of $2,100 without impacting service-level objectives . These trade-offs manifest differently across architectural layers—infrastructure decisions involve instance sizing and egress costs , platform decisions concern orchestration and observability , and application decisions address service abstraction and data management .

The evidence indicates that trade-off outcomes are highly context-dependent: financial services organizations achieve better results with centralized, integral architectures prioritizing consistency , while retail organizations benefit from decentralized, modular approaches prioritizing flexibility . Technical constraints including API differences, data gravity, and egress costs fundamentally constrain available options . Solution approaches include cloud-agnostic evaluation matrices , model-driven engineering methodologies , multi-objective optimization frameworks , and containerization with infrastructure-as-code standards . Organizations implementing these systematic approaches report 40% reductions in deployment inconsistencies, 60% faster incident resolution , and cross-provider application transfers within minutes without downtime . The findings demonstrate that while inherent tensions exist between competing architectural concerns, they are largely resolvable through context-aware frameworks that explicitly account for organizational priorities, workload characteristics, and regulatory constraints.

Paper search

We performed a semantic search using the query "Which architecture-level trade-offs shape the design and evaluation of enterprise architectures in technology-agnostic multi-cloud environments?" across over 138 million academic papers from the Elicit search engine, which includes all of Semantic Scholar and OpenAlex.

We retrieved the 499 papers most relevant to the query.

Screening

We screened in sources based on their abstracts that met these criteria:

- **Multi-cloud Environment Context**: Does the study explicitly address architectures designed for or deployed across multiple cloud providers (rather than focusing on a single cloud platform)?
- **Enterprise-scale Architectural Focus**: Does the study address architectural decisions at the enterprise level (rather than focusing solely on individual applications, microservices, or technical components)?

- **Architecture-level Trade-off Analysis**: Does the study examine trade-offs between architectural qualities such as performance, scalability, security, cost, vendor lock-in, complexity, or maintainability?
- **Technology-agnostic Architectural Approaches**: Does the study discuss architectural patterns, frameworks, or decisions that are not tied to specific vendor technologies or proprietary solutions?
- **Clear Methodology or Empirical Basis**: Does the study present transparent approaches, frameworks, methods, or empirical evaluations with clear methodological grounding (rather than lacking transparent methods or empirical basis)?
- **Publication Quality Standards**: Is the study a peer-reviewed academic publication or high-quality industry report from a reputable source (rather than an opinion piece, editorial, blog post, or non-peer-reviewed content)?

We considered all screening questions together and made a holistic judgement about whether to screen in each paper.

Data extraction

We asked a large language model to extract each data column below from each paper. We gave the model the extraction instructions shown below for each column.

- **Architecture Trade-offs**:

  Extract all architecture-level trade-offs explicitly discussed in the study, including:

- Specific trade-off dimensions (e.g., performance vs portability, flexibility vs complexity, cost vs capability)
- How each trade-off is characterized or defined
- Whether trade-offs are presented as inherent tensions or resolvable through specific approaches
- Any quantitative measures or qualitative assessments of trade-off severity
- Examples of how these trade-offs manifest in practice

- **Design Influence**:

  Extract evidence of how identified trade-offs shape architectural design decisions, including:

- Specific design patterns, frameworks, or approaches influenced by trade-offs
- Design principles or guidelines derived from trade-off considerations
- Architectural components or layers affected by trade-off decisions
- Design strategies used to manage or mitigate trade-offs
- Examples of design choices made to optimize specific trade-off dimensions

- **Evaluation Impact**:

  Extract information about how trade-offs influence architecture evaluation, including:

- Evaluation criteria or metrics shaped by trade-off considerations
- Assessment frameworks that account for trade-offs
- Methods for measuring or comparing trade-off outcomes
- How trade-offs affect architecture selection or comparison processes
- Evaluation challenges created by competing trade-off dimensions

- **Multi-cloud Constraints**:

  Extract specific characteristics of multi-cloud environments that create or influence trade-offs, including:

- Technical constraints (e.g., API differences, service incompatibilities, data transfer limitations)

- Operational challenges (e.g., management complexity, monitoring difficulties)
- Business constraints (e.g., vendor relationships, compliance requirements, cost models)
- Integration challenges between different cloud providers
- Technology-agnostic requirements and their implications

- **Contextual Factors**:

  Extract enterprise and environmental factors that influence trade-off decisions, including:

- Organization size, type, or industry sector
- Existing legacy systems and migration constraints
- Regulatory or compliance requirements
- Business objectives or strategic goals
- Resource constraints (budget, skills, time)
- Risk tolerance and governance models

- **Trade-off Outcomes**:

  Extract evidence about the consequences and outcomes of different trade-off decisions, including:

- Quantitative results (e.g., performance improvements, cost reductions, deployment times)
- Qualitative benefits or drawbacks observed
- Success/failure cases related to specific trade-off choices
- Lessons learned or recommendations about managing trade-offs
- Long-term implications of trade-off decisions
- Comparison of different approaches to handling the same trade-offs

- **Solution Approaches**:

  Extract proposed methods, frameworks, or techniques for managing architecture trade-offs, including:

- Specific methodologies or decision-making frameworks
- Tools or technologies designed to address trade-offs
- Architectural patterns that balance competing concerns
- Governance approaches for trade-off management
- Standards or best practices for multi-cloud architecture
- Innovation approaches that potentially resolve traditional trade-offs

Characteristics of Included Studies

The systematic review encompasses 25 sources examining architecture-level trade-offs in multi-cloud environments. These sources span from 2010 to 2025, representing the evolution of multi-cloud architectural thinking over fifteen years. The studies vary in their focus areas, methodological approaches, and industry contexts.

Study	Full Text Retrieved?	Study Type	Primary Focus Area
Bhargav Sai et al., 2025	No	Framework proposal	Decision-making frameworks for multi-cloud integration

Study	Full Text Retrieved?	Study Type	Primary Focus Area
G. Gibilisco et al., 2016	No	Methodology development	QoS-oriented design methodology with model-driven engineering
Antonis Papaioannou et al., 2013	No	Architecture proposal	Evaluation system for distributed application deployments
B. Yeganeh et al., 2023	No	Empirical study	Performance and cost characteristics of cloud backbones
Kishore Arul et al., 2022	No	Review	Data engineering challenges in multi-cloud environments
Antonis Papaioannou et al., 2013a	Yes	Architecture proposal	Long-term deployment history evaluation
Athanasia Evangelinou et al., 2015	No	Methodology development	Joint benchmark-analytic approach for design-time assessment
Phani Sekhar Emmanni et al., 2020	No	Framework proposal	Cloud-native application architecture for multi-cloud
Karwan Jameel Merseedi et al., 2024	Yes	Review	Hybrid and federated cloud architectures
Stefano Berlato et al., 2022	No	Methodology development	Cryptographic access control architecture optimization
Prem Nishanth Kothandarama et al., 2025	No	Review	Cross-cloud portability and developer platforms
Stefano Berlato et al., 2020	No	Methodology development	CAC architecture exploration and optimization
Pooyan Jamshidi et al., 2015	Yes	Pattern catalogue	Cloud migration patterns for multi-cloud
Nicolas Ferry et al., 2013	Yes	Framework proposal	Model-driven provisioning and deployment
Mohammad Y. Hajjat et al., 2010	Yes	Optimization model	Enterprise application migration planning
Ruiqing Cao et al., 2022	No	Empirical study	Enterprise data architecture and cloud adoption
Manish Ahuja et al., 2022	No	Framework proposal	Edge-aware multi-cloud hybrid architecture
Peter-Christian Quint et al., 2018	Yes	DSL development	Multi-cloud DSL for elastic applications

Study	Full Text Retrieved?	Study Type	Primary Focus Area
Heiko Koziolek et al., 2011	No	Architectural style	SPOSAD style for multi-tenant applications
V. Casola et al., 2018	Yes	Optimization approach	Security-by-design in multi-cloud
N. Joukov et al., 2013	No	Policy framework	Cloud interoperability via application rebuilds
K. Ramchand et al., 2017	No	Decision framework	Cloud architectural decision framework
Haidong Zhao et al., 2022	Yes	System development	Multi-cloud serverless computing support
Georgios Chatzithanasis et al., 2021	Yes	Efficiency analysis	Cost-efficient bundling in multi-cloud
Praveen Kumar Reddy Gujjala et al., 2023	No	Framework proposal	Cloud-native lakehouses with serverless strategies

The studies represent diverse methodological approaches including framework proposals, empirical investigations, optimization models, and architectural pattern development. Eight studies provided full text access, while seventeen were analyzed based on abstracts. The temporal distribution shows continued scholarly interest in multi-cloud trade-offs, with particular intensity in recent years (2020-2025).

Architecture-Level Trade-offs in Multi-Cloud Environments

Primary Trade-off Dimensions

The analysis reveals seven primary trade-off dimensions that consistently shape multi-cloud architecture decisions across the literature.

Trade-off Dimension	Studies Identifying	Characterization	Resolvability
Performance vs. Portability	12 studies	Designing for cross-platform operation reduces optimization for specific providers	Partially resolvable through abstraction layers
Cost vs. Capability	15 studies	Higher capability often requires increased expenditure on resources or services	Resolvable through optimization frameworks
Flexibility vs. Complexity	14 studies	Greater architectural flexibility introduces management and integration complexity	Resolvable through standardization
Security vs. Usability	4 studies	Enhanced security mechanisms may impede operational usability	Resolvable through optimization

Trade-off Dimension	Studies Identifying	Characterization	Resolvability
Vendor Lock-in vs. Scalability	6 studies	Avoiding vendor dependency may limit access to provider-specific scaling features	Partially resolvable through DSLs
Modularity vs. Integration	3 studies	Modular designs offer flexibility but challenge consistent integration	Context-dependent resolution
Consistency vs. Complexity	4 studies	Ensuring consistent behavior across clouds increases architectural complexity	Resolvable through IaC standards

The cost versus capability trade-off emerges as the most frequently discussed dimension, with quantitative evidence demonstrating cost reductions of 39% to 78% achievable through systematic optimization approaches . Similarly, annual savings of $2,100 were demonstrated without impacting service-level objectives through careful deployment selection .

Trade-off Manifestations by Architectural Layer

Trade-offs manifest differently across architectural layers. At the infrastructure layer, heterogeneity in cloud solutions creates barriers to seamless operations , while API differences and service incompatibilities introduce technical constraints . At the platform layer, the challenge of achieving unified observability across providers creates operational difficulties . At the application layer, performance variability due to congestion and provider policies affects service delivery .

Architectural Layer	Primary Trade-offs	Key Manifestations
Infrastructure	Cost vs. capability, vendor lock-in vs. scalability	Instance sizing decisions, provider selection , egress costs
Platform	Flexibility vs. complexity, consistency vs. complexity	Orchestration platform choices, IaC standardization
Application	Performance vs. portability, security vs. usability	Service abstraction design, data management strategies
Data	Modularity vs. integration, cost vs. capability	Data architecture design (modular vs. integral) , data gravity challenges

Design Influence of Trade-offs

Design Patterns and Frameworks

Trade-off considerations have driven the development of numerous architectural patterns and design approaches. Cloud-agnostic evaluation matrices enable systematic comparison of provider capabilities across workload require-

ments, service compatibility, compliance standards, network performance, and cost metrics . Infrastructure-as-Code standards ensure consistency while reducing security risks across diverse cloud environments .

Design Pattern/Framework	Trade-offs Addressed	Implementation Approach
Abstraction layers (CCIM, CPIM, CPSM)	Performance vs. portability	Three-level abstraction from computation-independent to provider-specific
Federated integration architectures	Flexibility vs. complexity	Event-driven patterns connecting legacy and modern systems
Containerization and microservices	Cost vs. capability, portability	Docker, Kubernetes for deployment agility
Service mesh	Consistency vs. complexity	Interoperability and system consistency
Data lakehouse architectures	Flexibility vs. performance	Delta Lake, Snowflake combining lake flexibility with warehouse performance
Security-by-design	Security vs. cost	AHP-based optimization for security-cost balance

The model-driven engineering approach addresses trade-offs by enabling assessment of expected QoS early in design stages . High-level architectural models are transformed into Layered Queuing Network performance models that can be analyzed with solvers to derive performance metrics . This allows software architects to refine design choices based on quantitative trade-off analysis.

Design Principles Derived from Trade-off Analysis

Several design principles emerge from trade-off considerations. The separation of platform definition from application definition enables multi-cloud transferability at runtime . Architectural abstraction combined with developer experience, data management, performance engineering, and security integration provides a multidimensional framework for addressing cross-cloud requirements .

For data architecture specifically, the choice between modular (microservices) and integral (single fabric) designs depends on organizational context. Retail establishments benefit from decentralized approaches ensuring flexible adaptability to external market environments, while financial services establishments benefit from centralized approaches ensuring consistent integration of internal data sources and robust control .

Evaluation Impact of Trade-offs

Evaluation Criteria and Metrics

Trade-off considerations shape the evaluation criteria used for multi-cloud architecture assessment. Performance metrics such as availability and response time are balanced against operational costs . The evaluation extends to path, delay, and traffic-cost characteristics when assessing multi-cloud overlay configurations .

Evaluation Dimension	Metrics/Criteria	Trade-off Context
Performance	Response time, latency, availability	Balanced against cost and portability
Cost	Operational costs, egress fees, duration fees	Balanced against capability and performance
Quality of Service	SLAs, consistency guarantees, fault tolerance	Balanced against flexibility and cost
Security	Security coverage scores	Balanced against cost and usability
Portability	Cross-cloud deployment capability	Balanced against performance optimization

Assessment Frameworks

Multiple assessment frameworks have been developed to account for trade-offs in architecture evaluation. The Analytic Hierarchy Process (AHP) methodology enables evaluation of trade-offs between security and cost by assigning priorities to different criteria based on developer preferences . Multi-objective Combinatorial Optimization Problem (MOCOP) frameworks allow exploration of different architectures and their implications through "What-if" analysis .

The Data Envelopment Analysis (DEA) methodology assesses efficiency of multi-cloud solutions, focusing on trade-offs between cost, performance, and scalability . This approach categorizes services and highlights trade-offs in resource allocation while handling the complexity of multi-cloud environments .

Evaluation Challenges

Competing trade-off dimensions create significant evaluation challenges. The complexity of finding optimal deployment configurations has been shown to be NP-hard, necessitating heuristic approaches . Varying transit costs and pricing models across geographic regions complicate performance and cost evaluation . Multi-tenant environments with varying performance due to congestion and provider policies add temporal variability to evaluation outcomes .

Multi-Cloud Environmental Constraints

Technical Constraints

Technical constraints in multi-cloud environments fundamentally shape trade-off decisions. API differences and service incompatibilities between providers create integration challenges . Data transfer limitations and egress costs significantly impact architectural decisions . The heterogeneity in cloud infrastructure means virtual resources often have only nominal or indicative characterizations .

Constraint Category	Specific Constraints	Impact on Trade-offs
Technical	API differences , service incompatibilities , data transfer limitations	Forces portability vs. performance decisions

Constraint Category	Specific Constraints	Impact on Trade-offs
Operational	Management complexity , monitoring difficulties , governance fragmentation	Increases flexibility vs. complexity tension
Business	Vendor relationships , cost models , compliance requirements	Constrains cost vs. capability options
Integration	Legacy system connectivity , interoperability standards	Affects consistency vs. complexity balance

Data gravity emerges as a particularly significant constraint, where large data volumes create dependencies on specific providers due to expensive data egress fees . This constraint directly influences vendor lock-in versus flexibility trade-offs.

Operational and Business Constraints

Operational challenges include fragmented governance protocols and difficulties achieving unified observability across cloud boundaries . The lack of well-integrated feedback loops for monitoring and SLA assessments complicates continuous evaluation .

Business constraints encompass regulatory compliance requirements , vendor lock-in risks , and varying pricing models across providers and regions . Organizations must balance these constraints against capability requirements, with the on-demand leasing model complicating resource allocation and scheduling decisions .

Contextual Factors Influencing Trade-off Decisions

Industry and Organizational Context

Industry sector significantly influences trade-off priorities and decisions. Implementation evidence spans fintech, pharmaceutical, healthcare, retail, and manufacturing organizations . Financial services organizations prioritize consistent integration of internal data sources and robust control through centralized architectures , while retail organizations prioritize flexible adaptability through decentralized approaches .

Industry Sector	Primary Trade-off Priorities	Architectural Preference
Financial services	Security vs. usability, consistency vs. flexibility	Centralized, integral architecture
Retail	Flexibility vs. complexity, cost vs. capability	Decentralized, modular architecture
Healthcare	Compliance vs. flexibility, privacy vs. efficiency	Hybrid with on-premise components
Manufacturing	Performance vs. portability, cost vs. capability	Industry-specific optimization

Large enterprises face particular challenges with enterprise software and middleware non-interoperability adding

complexity beyond cloud provider issues . The complexity of managing data centers with large numbers of applications creates significant migration constraints .

Regulatory and Compliance Requirements

Regulatory requirements fundamentally constrain trade-off options. Country regulations may limit data storage options, forcing specific architectural decisions . Hard regulatory policy constraints and privacy requirements necessitate on-premise processing of certain data . Industry-specific regulations and national privacy laws must be accommodated in migration planning .

Resource Constraints

Budget constraints drive cost optimization efforts, with organizations seeking to convert capital expenditure to operational costs while addressing inefficient infrastructure use . Skills constraints influence technology selection, with the need for technology-independent perspectives to enable wider adoption of multi-tenant architectures . Time constraints affect migration planning, with organizations balancing rapid deployment against thorough architectural assessment .

Trade-off Outcomes and Consequences

Quantitative Outcomes

Systematic management of trade-offs yields measurable benefits across multiple dimensions.

Outcome Measure	Quantitative Result	Context
Deployment consistency	40% reduction in inconsistencies	Framework implementation across industries
Incident resolution	60% faster resolution	Observability solutions implementation
Cost optimization	25% improvement	PACE layering model application
Cost reduction	39% to 78% reduction vs. best practices	Heuristic optimization approach
Annual savings	$2,100 per year	Cost-effective deployment selection
Migration cost reduction	Order of magnitude reduction	Standardization and documentation policies
Multi-cloud efficiency	14-19.2% improvement	DEA-evaluated bundling strategies
Transfer time	Minutes without downtime	Cross-provider application transfer

Cost savings demonstrate significant variation based on deployment approach. Compute-class server migration yields savings of $1,577 per year, while storage-class servers save $17,280 per year, though with trade-offs in delay (17% increase in mean, 12% increase in variance) .

Qualitative Outcomes

Qualitative benefits include improved operational efficiency and business agility through coordinated ecosystem orchestration . Organizations achieve better risk management and avoidance of vendor lock-in through multi-provider strategies . Enhanced flexibility in decision-making enables partial migration approaches that balance privacy, performance, and cost considerations .

Qualitative drawbacks observed include increased complexity in managing multiple providers , potential security concerns when avoiding vendor lock-in through on-premise proxy deployment , and challenges with stateful application migration involving large datasets . Project delays, budget overruns, and technical debt creation occur when trade-offs are not properly assessed during migration planning .

Context-Dependent Outcomes

Trade-off outcomes vary significantly by context. AWS Lambda outperforms other providers in billing duration and latency for image processing workloads , while other providers may be preferable for different workload types. The effectiveness of modular versus integral data architectures depends on whether the organization operates in retail (favoring modularity) or financial services (favoring integration) .

Solution Approaches for Managing Trade-offs

Methodologies and Decision Frameworks

Multiple methodologies have been proposed for systematic trade-off management in multi-cloud environments.

Methodology/Framework	Purpose	Key Features
Cloud-agnostic evaluation matrices	Provider comparison	Compare capabilities across workload requirements, compliance, performance, cost
Model-driven engineering (Palladio, MODACloudML)	Design-time QoS assessment	Transform models to performance models for analysis
MOCOP solvers	Architecture optimization	Multi-objective optimization for heterogeneous requirements
AHP methodology	Security-cost balancing	Multiple criteria decision-making with weighted priorities
DEA methodology	Efficiency evaluation	Assess multi-provider bundling efficiency
Assembly-based migration	Flexible migration planning	Reusable, composable migration patterns

The PACE layering model provides systematic prioritization of modernization efforts , while joint benchmarking and optimization methodologies support legacy application migration with minimum cost and QoS guarantees .

Tools and Technologies

Specific tools and technologies address trade-off management across the architectural stack. Container orchestration platforms (Kubernetes, Apache Airflow) enable workload portability . Infrastructure as Code tools (Terraform) provide automated, consistent infrastructure provisioning . Service mesh technologies enable interoperability and system consistency across cloud boundaries .

Data management technologies include metadata registries (Apache Atlas) for governance , data lakehouse architectures (Delta Lake, Snowflake) combining flexibility with performance , and federated query engines for cross-cloud data access . For serverless environments, multi-cloud libraries provide common interfaces across FaaS and BaaS offerings from different providers .

Architectural Patterns

Several architectural patterns balance competing concerns in multi-cloud environments. The three-level abstraction model (CCIM, CPIM, CPSM) enables portability while allowing provider-specific optimization when needed . Security-by-design processes incorporate multiple criteria decision-making through functional design, security design, optimal deployment identification, and security enforcement phases .

Fine-grained service-based cloud architecture migration patterns provide reusable building blocks for migration planning . The SPOSAD architectural style addresses multi-tenant software scalability and customization requirements . Domain-specific languages separate elastic platform definition from cloud application definition, enabling runtime multi-cloud adaptation .

Governance and Standards

Governance approaches include policy-based management for addressing challenges like stateful application migration, data gravity, and compliance . Standards such as TOSCA and Cloudify enable automated deployment across multiple cloud service providers . Stricter enterprise application development and maintenance policies, combined with documentation standardization, enable predictable and cost-effective application transformations .

Synthesis

Reconciling Heterogeneous Findings

The literature presents apparent contradictions regarding optimal approaches to multi-cloud trade-off management. Some studies emphasize the benefits of abstraction and standardization for portability , while others highlight performance penalties associated with these approaches . This heterogeneity can be explained through several analytical lenses.

Context and Population Distinctions : The effectiveness of specific trade-off management strategies depends critically on organizational context. Financial services organizations achieve better outcomes with integral, centralized data architectures that prioritize consistency and control , while retail organizations benefit from modular, decentralized approaches prioritizing flexibility . Both findings are valid within their respective domains—the apparent contradiction reflects different optimization targets rather than conflicting evidence.

Study Quality and Methodology Considerations : Studies providing quantitative evidence of trade-off outcomes (cost reductions of 39-78% , annual savings of $2,100 , efficiency improvements of 14-19.2%) employed systematic optimization methodologies and empirical validation. Studies making broader claims about trade-off characteristics

without quantitative validation should be weighted accordingly. The industrial case study validations and real-world implementations across multiple industries provide stronger evidence than theoretical frameworks alone.

Mechanistic Explanations : The vendor lock-in versus scalability trade-off manifests differently depending on the architectural layer under consideration. At the infrastructure layer, vendor lock-in primarily affects egress costs and API compatibility . At the platform layer, it affects orchestration and monitoring capabilities . At the application layer, it affects service abstraction and data management . Solutions effective at one layer (e.g., containerization for application portability) may not address constraints at other layers (e.g., data gravity at the storage layer).

Dose-Response Relationships : The relationship between abstraction level and performance overhead appears non-linear. The three-level abstraction model (CCIM, CPIM, CPSM) suggests that computation-independent models incur minimal overhead while enabling maximum portability, provider-independent models add moderate specificity with moderate overhead, and provider-specific models maximize performance at the cost of portability. Organizations should select abstraction levels matching their actual portability requirements rather than defaulting to maximum abstraction.

Consolidated Findings

For organizations prioritizing **cost optimization** , systematic application of optimization frameworks yields reductions of 39-78% compared to provider best practices , with heuristic approaches enabling solution identification in 16-40 minutes even for complex scenarios . Annual savings of $2,100 are achievable without impacting service-level objectives through careful deployment selection .

For organizations prioritizing **operational consistency** , Infrastructure-as-Code standards and comprehensive observability solutions reduce deployment inconsistencies by 40% and improve incident resolution times by 60% . Standardization and documentation policies enable order-of-magnitude cost reductions in application migrations .

For organizations prioritizing **flexibility and portability** , containerization and microservices architectures enable cross-provider application transfer within minutes without downtime . However, stateful applications with large datasets remain challenging to migrate , and data gravity effects constrain practical portability options .

For organizations in **regulated industries** , the security versus usability trade-off can be managed through MOCOP-based optimization that accounts for heterogeneous trust assumptions and requirements . The security-by-design approach with AHP methodology enables explicit balancing of security coverage against cost based on organizational priorities .

The evidence consistently indicates that trade-off management requires explicit attention to organizational context, workload characteristics, and strategic priorities. Generic solutions are less effective than context-aware approaches that acknowledge the inherent tensions in multi-cloud architecture while providing systematic methods for navigating them.

References

Antonis Papaioannou. “Design and Implementation of a System for Evaluating Distributed Application Deployments in Multi-Clouds,” 2013.

Antonis Papaioannou, and K. Magoutis. “An Architecture for Evaluating Distributed Application Deployments in Multi-Clouds.” *IEEE International Conference on Cloud Computing Technology and Science*, 2013.

Athanasia Evangelinou, M. Ciavotta, George Kousiouris, and D. Ardagna. “A Joint Benchmark-Analytic Approach For Design-Time Assessment of Multi-Cloud Applications.” *International Conference on Cloud Forward*, 2015.

B. Yeganeh, Ramakrishnan Durairajan, R. Rejaie, and W. Willinger. "A Case for Performance- and Cost-Aware Multi-Cloud Overlays." *IEEE International Conference on Cloud Computing*, 2023.

Bhargav Sai, and Pillati. "Leading Through Complexity: Decision-Making Frameworks for Multi-Cloud Integration and Platform Scalability." *Journal of Information Systems Engineering & Management*, 2025.

G. Gibilisco. "A Methodology and a Tool for QoS-Oriented Design of Multi-Cloud Applications," 2016.

Georgios Chatzithanasis, Evangelia Filiopoulou, C. Michalakelis, and M. Nikolaidou. "Exploring Cost-Efficient Bundling in a Multi-Cloud Environment." *Simulation Modelling Practice and Theory*, 2021.

Haidong Zhao, Zakaria Benomar, Tobias Pfandzelter, and N. Georgantas. "Supporting Multi-Cloud in Serverless Computing." *International Conference on Utility and Cloud Computing*, 2022.

Heiko Koziolek. "The SPOSAD Architectural Style for Multi-Tenant Software Applications." *2011 Ninth Working IEEE/IFIP Conference on Software Architecture*, 2011.

K. Ramchand, Mohan Baruwal Chhetri, and R. Kowalczyk. "Towards a Flexible Cloud Architectural Decision Framework for Diverse Application Architectures." *ACIS*, 2017.

Karwan Jameel Merseedi, and Dr. Subhi R. M. Zeebaree. "Cloud Architectures for Distributed Multi-Cloud Computing: A Review of Hybrid and Federated Cloud Environment." *Indonesian Journal of Computer Science*, 2024.

Kishore Arul. "Data Engineering Challenges in Multi-Cloud Environments: Strategies for Efficient Big Data Integration and Analytics." *International Journal of Scientific Research and Management*, 2022.

Manish Ahuja, Narendranath Sukhavasi, Swapnajeet Choudhury, Kalpataru Das, Kapil Singi, Kuntal Dey, and Vikrant S. Kaulgud. "MCDA Framework for Edge-Aware Multi-Cloud Hybrid Architecture Recommendation." *International Conference on Automated Software Engineering*, 2022.

Mohammad Y. Hajjat, Xin Sun, Y. Sung, D. Maltz, Sanjay G. Rao, K. Sripanidkulchai, and Mohit Tawarmalani. "Cloudward Bound: Planning for Beneficial Migration of Enterprise Applications to the Cloud." *Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, 2010.

N. Joukov, and V. Shorokhov. "Cloud Interoperability via Quick Enterprise Applications Re-Builds." *International Conference on Cloud Computing and Services Science*, 2013.

Nicolas Ferry, A. Rossini, Franck Chauvel, Brice Morin, and Arnor Solberg. "Towards Model-Driven Provisioning, Deployment, Monitoring, and Adaptation of Multi-Cloud Systems." *2013 IEEE Sixth International Conference on Cloud Computing*, 2013.

Peter-Christian Quint, and Nane Kratzke. "Towards a Lightweight Multi-Cloud DSL for Elastic and Transferable Cloud-Native Applications." *International Conference on Cloud Computing and Services Science*, 2018.

Phani Sekhar Emmanni. "Architecting Cloud-Native Applications for Multi-Cloud Environments." *International Journal of Science and Research (IJSR)*, 2020.

Pooyan Jamshidi, C. Pahl, Samuel J. Chinenyeze, and Xiaodong Liu. "Cloud Migration Patterns: A Multi-Cloud Service Architecture Perspective." *ICSOC Workshops*, 2015.

Praveen Kumar Reddy Gujjala. "The Future of Cloud-Native Lakehouses: Leveraging Serverless and Multi-Cloud Strategies for Data Flexibility." *International Journal of Scientific Research in Computer Science Engineering and Information Technology*, 2023.

Prem Nishanth Kothandarama. "Designing Developer Platforms for Cross-Cloud Portability and Scale." *International Research Journal on Advanced Engineering Hub (IRJAEH)*, 2025.

Ruiqing Cao, and M. Iansiti. "Cloud Adoption and Digital Transformation: The Paradoxical Role of Enterprise Data Architecture." *Social Science Research Network*, 2022.

Stefano Berlato, R. Carbone, Adam J. Lee, and Silvio Ranise. "Exploring Architectures for Cryptographic Access Control Enforcement in the Cloud for Fun and Optimization." *ACM Asia Conference on Computer and Communications Security*, 2020.

———. "Formal Modelling and Automated Trade-Off Analysis of Enforcement Architectures for Cryptographic Access Control in the Cloud." *ACM Transactions on Privacy and Security*, 2022.

V. Casola, Alessandra De Benedictis, M. Rak, and Umberto Villano. "Security-by-Design in Multi-Cloud Applications: An Optimization Approach." *Information Sciences*, 2018.