HAW HAMBURG

MASTER THESIS
Leonard Vincent Simon Pahlke

Measuring Cloud Native Sustainability

FAKULTÄT TECHNIK UND INFORMATIK
Department Informatik

Faculty of Engineering and Computer Science
Department Computer Science

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Leonard Vincent Simon Pahlke

Measuring Cloud Native Sustainability

Masterarbeit eingereicht im Rahmen der Masterprüfung
im Studiengang *Master of Science Informatik*
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Stefan Sarstedt
Zweitgutachter: Prof. Dr. Olaf Zukunft

Eingereicht am: 12. Juni 2024

Leonard Vincent Simon Pahlke

Thema der Arbeit

Measuring Cloud Native Sustainability

Stichworte

Cloud, Cloud Native, Grüne Softwareentwicklung, Cloud Native Sustainability, Observability, Metriken

Kurzzusammenfassung

Nachhaltigkeit und Digitalisierung sind zwei zentrale Transformationen unserer Gesellschaft. Diese Arbeit befasst sich mit der Schnittmenge dieser Themenfelder und stellt Carbonaut vor, ein Open-Source-Projekt zur Sammlung und Verarbeitung von Nachhaltigkeitsmetriken in Cloud-Umgebungen. Carbonaut ermöglicht die effiziente Sammlung von Metriken, um den Nachhaltigkeitsfußabdruck eines Cloud-Systems zu bestimmen. Eine systematische Bewertung des Projekts zeigt, dass Carbonaut einen wertvollen Beitrag zur Verbesserung des Verständnisses der Nachhaltigkeit in Cloud-Umgebungen leisten kann. Die Arbeit leistet einen Beitrag zum Verständnis von grüner Softwareentwicklung mit einem Fokus auf Cloud und Cloud-Native Sustainability. Sie regt die Entwicklung neuer innovativer Projekte in diesem Bereich an.

Leonard Vincent Simon Pahlke

Title of Thesis

Measuring Cloud Native Sustainability

Keywords

Cloud, Cloud Native, Sustainable Software Engineering, Cloud Native Sustainability, Observability, Metrics

Abstract

Sustainability and digitalization are two key transformations of our society. This thesis investigates the intersection of these topics and introduces Carbonaut, an open source

project for collecting and processing sustainability metrics in cloud environments. Carbonaut enables the efficient collection of metrics to determine the sustainability footprint of a cloud system. A systematic evaluation of the project shows that Carbonaut can make a valuable contribution to improving the understanding of sustainability in cloud environments. The work contributes to the understanding of green software engineering with a focus on cloud and cloud-native sustainability. It engages the development of new innovative projects in this area.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

1.1 Motivation

Nachhaltigkeit ist ein Thema, das unsere Gesellschaft prägt und dieses Jahrhundert voraussichtlich als treibendes Motiv maßgeblich verändern wird. Softwareentwicklung und Digitalisierung schließen sich dem als weitere Transformationsmotoren an. Diese Arbeit befasst sich mit diesen beiden Transformationen und sucht nach Schnittstellen, an denen sie zusammenkommen. Denn wenn beide Trends über Jahrzehnte oder länger wirken, müssen sie auch gemeinsam gedacht werden.

Ein weiterer Gedanke dieser Arbeit wurzelt in der Kuriosität, besser zu verstehen, welche Ressourcen Software auf Systemebene bindet und wie dies erfahrbar gemacht werden kann. Hierbei gibt es verschiedene Ansatzmöglichkeiten. Diese Arbeit betrachtet dies als technisches Problem, das in der Cloud angesiedelt ist und wo neue Metriken, Schnittstellen und Datenmodelle als Lösungen entwickelt werden.

1.2 Ziele

Ziel dieser Arbeit ist es, eine Brücke zwischen Nachhaltigkeit und Digitalisierung zu schlagen, um eine Integration zu ermöglichen. Es soll gezeigt werden, wie Nachhaltigkeitsforschung die Softwareentwicklung beeinflusst und welche Themen dabei entstanden sind und entstehen. Konkret soll ein Projekt entwickelt werden, das einen Beitrag dazu leistet, Metriken zur Nachhaltigkeit in einem Cloud-Kontext zu verarbeiten, zu integrieren und miteinander zu kombinieren. Nach der Entwicklung und Bewertung des Projekts sollen Perspektiven für die Weiterentwicklung aufgezeigt werden, an die zukünftige Forschungsarbeiten anknüpfen können.

1.3 Gliederung

Diese Arbeit lässt sich in zwei Abschnitte unterteilen. Die Kapitel 2 und 3 befassen sich mit den theoretischen Grundlagen zur Nachhaltigkeit, Digitalisierung und Grünen Softwareentwicklung. Die Kapitel 4, 5 und 6 erarbeiten das Projekt Carbonaut, analysieren und bewerten dieses. Die Kapitel greifen ineinander über und bauen aufeinander auf.

In Kapitel 2 werden die beiden Themenfelder Nachhaltigkeit und Digitalisierung systematisch untersucht. Nachhaltigkeit wird dabei knapp aufbereitet. Dabei liegt ein Fokus auf der Betrachtung der thematischen Schnittmengen. Nachhaltigkeit und Digitalisierung beeinflussen einander. Auf der Softwareseite wird dies mit dem Forschungsfeld der nachhaltigen Softwareentwicklung abgebildet, welches auch unter grüner Softwareentwicklung verstanden wird. Diesem Forschungsgebiet wird sich Kapitel drei widmen.

Das Kapitel 3 startet mit einer Diskussion über Forschungsarbeiten zu grüner Softwareentwicklung. Anschließend wird in das Thema der Cloud und Cloud-Native überführt, was den ersten Teil abschließt.

Im zweiten Teil wird das Projekt Carbonaut bearbeitet. Das Projekt wird in Kapitel 4 mithilfe einer Architekturdokumentation detailliert beschrieben und umgesetzt. Das Carbonaut-Projekt ist ein Open-Source-Projekt zur Akkumulation von Metriken zur Bestimmung des Nachhaltigkeitsfußabdrucks eines Cloud-Systems. Carbonaut kann als Baustein eines Cloud-nativen Systems verstanden werden.

Das Projekt wird in Kapitel 5 mithilfe von Test-Szenarien untersucht und beobachtet. Das Ziel dieses Abschnitts ist es, die Vorarbeit zu leisten, um die Güte des Carbonaut-Projekts zu validieren. Diese Analyse folgt in Kapitel 6. Die Analyse bespricht alle aufgestellten Qualitätsziele und zeigt Vorschläge auf, wie Carbonaut weiterentwickelt werden kann.

Die Arbeit schließt mit einer Zusammenfassung und einem Ausblick ab.

2 Nachhaltigkeit und Digitalisierung

In diesem Kapitel werden die theoretischen Grundlagen der Nachhaltigkeit sowie deren Zusammenhang mit der Digitalisierung beschrieben und erklärt. Dabei wird insbesondere die Bedeutung der Nachhaltigkeit in der Softwareentwicklung hervorgehoben.

2.1 Nachhaltigkeit

„Nachhaltigkeit“ ist als Begriff im 18. Jahrhundert aufgekommen, als Hans Carl von Carlowitz erkannte, dass die damals zunehmende Abholzung der Wälder in Europa in absehbarer Zukunft zu einem Engpass führen werde und es Lösungen benötigt, nachhaltig mit der Ressource Holz umzugehen[68]. Es wurde daraufhin die Aufforstung als Maßnahme abgeleitet, um dem absehbaren Holzmangel entgegenzuwirken.

Mit dem Aufschwung der globalen Wirtschaft im 20. und 21. Jahrhundert entstand großer Wohlstand, vorwiegend in Industrieländern, und verbesserte das tägliche Leben[3]. Um diesen Wohlstand zu generieren, wurde in die Natur verstärkt eingegriffen. Straßen wurden beispielsweise gebaut, natürliche Ressourcen wurden abgebaut, Pestizide eingesetzt und Unterwasser-Seekabel verlegt und so weiter. Der Nobelpreisträger Paul Crutzen et al. beschreibt diese Zeit als »Das Anthropozän«[18][19]. Die Art und Weise, wie wir leben, produzieren und konsumieren, hat zu einer kritischen Phase in der Geschichte unseres Planeten geführt. Diese ist gekennzeichnet durch dramatische Veränderungen im Klima, der Biodiversität und von geologischen Prozessen.

Wie in dem Buch »Die Gaia-Hypothese«[40] postuliert wird, kann die Erde als ein zusammenhängendes komplexes System verstanden werden. Es ist demnach logisch, dass es zu *Komplikationen* kommt, wenn *extern durch die menschliche Hand* in dieses System *künstlich* eingegriffen wird. Dies wird in 2.1.2 ausgeführt. Im Laufe des zwanzigsten

Jahrhunderts wurde Nachhaltigkeit als Herausforderung erkannt[12][22] und viele wegweisende Arbeiten veröffentlicht, die versuchen, das Problem besser zu fassen und Lösungen zu entwickeln.

2.1.1 Segmentierung der Nachhaltigkeit

Nachhaltigkeit wird mittlerweile in vielen verschiedenen Kontexten diskutiert. Daher ist es wichtig, einen Überblick über die Kontexte, die Segmentierung der Nachhaltigkeit, zu bekommen. Es werden zunächst die Aspekte der Nachhaltigkeit und anschließend Umweltsphären vorgestellt.

Nachhaltigkeitsaspekte

1987 wurde Nachhaltigkeit erstmals als Leitthema breit auf politisch internationaler Bühne diskutiert[67]. Der Abschlussbericht »Our Common Future« definierte den Begriff Nachhaltigkeit genauer als die Fähigkeit zukünftiger Generationen, ihre eigenen Bedürfnisse zu befriedigen, ohne die Möglichkeiten künftiger Generationen zu beeinträchtigen. Nachhaltigkeit wird zudem in drei Aspekte: ökologische, soziale und ökonomische Nachhaltigkeit unterteilt. Diese drei Aspekte müssen bei der Entwicklung von Maßnahmen in Einklang gebracht werden. In dem nachfolgenden Diagramm 2.1 sind diese drei Aspekte dargestellt.

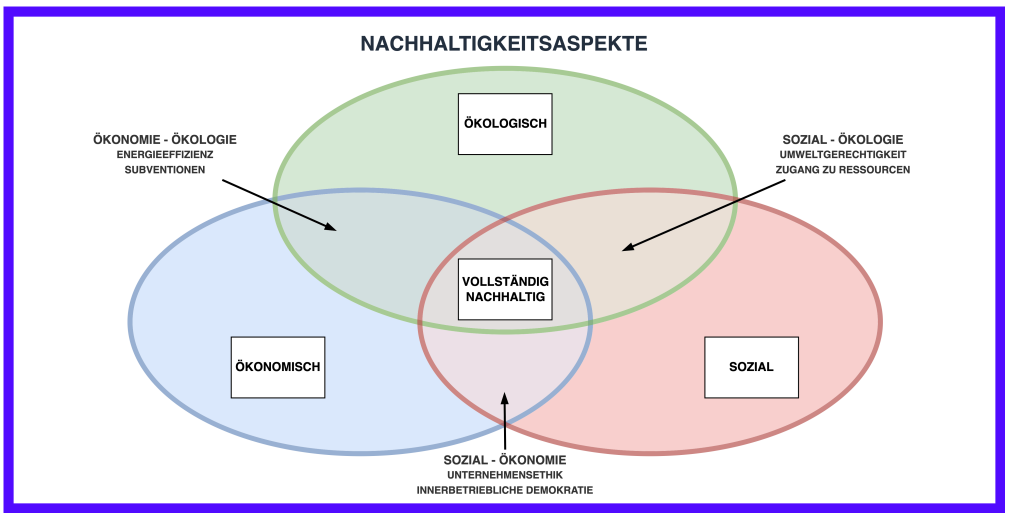

Abbildung 2.1: Nachhaltigkeitsaspekte[67]

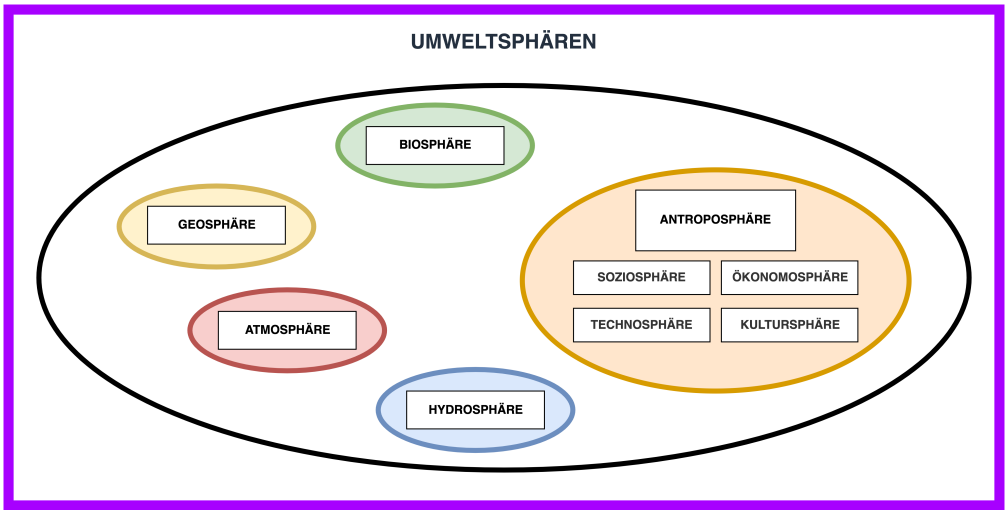

Abbildung 2.2: Umweltsphären

In 2.1 werden die drei Aspekte der Nachhaltigkeit – sozial, ökologisch und ökonomisch – visualisiert. Das Diagramm zeigt auch die Schnittmengen zwischen diesen Aspekten auf. Ein Beispiel für die sozial-ökologische Schnittmenge ist die Umweltgerechtigkeit, was unter anderem den Zugang zu Wasser beinhaltet. Das Streben nach nachhaltiger Entwicklung erfordert die Berücksichtigung aller drei Aspekte, was im Zentrum des Diagramms veranschaulicht wird.

Diese Aspekte der Nachhaltigkeit können erweitert werden. Dabei werden gelegentlich individuelle und technische Nachhaltigkeit vorgeschlagen[5]. Individuelle Nachhaltigkeit wird exemplarisch am Gedanken des Minimalismus beschrieben, ist jedoch facettenreicher. Im Sinne dieser Arbeit ist sie jedoch nicht weiter von zentraler Bedeutung. Technische Nachhaltigkeit wird in den kommenden zwei Unterkapiteln 2.2 und 3.1 erläutert.

Minimalismus beschreibt einen Denkansatz, bei dem sich auf das Wesentliche beschränkt wird. Der Begriff ist damit positiv konnotiert. Minimalismus kann als Lebensphilosophie, Tugend oder pragmatisches Motiv aufgefasst werden und wird dem Feld der Ethik zugeordnet. Er kann sich auf das Tun und Handeln oder den Umgang mit materialistischen und immateriellen Gegenständen beziehen. Es geht um bewusstes Tun, speziell um bewusstes wenig Tun oder Haben, und um bewusstes wenig Haben. Er kann damit auch als Kritik an unüberlegtem Konsumieren aufgefasst werden. Minimalismus wird relevant, wenn zu viel von etwas vorhanden ist und eine Rückbesinnung auf den Kern erforderlich wird. Was dabei als „das Wenige“ gilt, hängt vom jeweiligen Kontext ab, der je nach Hintergrund unterschiedlich ist. Das wenig Haben soll zu weniger Verpflichtungen und mehr

Freiheit führen. Minimalismus findet sich, wie in 3.1 später beschrieben, in der Softwareentwicklung im Trend um *Lean* Software-Engineering wieder, aber auch im allgemeinen Kontext der nachhaltigen Softwareentwicklung.

Umweltsphären

Eine weitere Möglichkeit, Nachhaltigkeit zu betrachten, ist die Segmentierung in Umweltsphären. Diese Unterteilung hat sich im Laufe der Zeit durch wissenschaftliche Arbeiten als sinnvoll ergeben [20][63]. Das Diagramm 2.2 zeigt die fünf Umweltsphären: Biosphäre, Geosphäre, Hydrosphäre, Atmosphäre und Anthroposphäre. Die Anthroposphäre ist zudem weiter unterteilt, da hier spätere Diskussionen über nachhaltige Software stattfinden werden.

- **Biosphäre**: der Lebensraum aller Organismen auf der Erde, mit Schwerpunkt auf Artenvielfalt, Waldschutz und nachhaltiger Landwirtschaft.
- **Geosphäre**: die festen Bestandteile der Erde, mit Schwerpunkt auf nachhaltigem Umgang mit natürlichen Ressourcen und Bodenschutz.
- **Hydrosphäre**: alle Wasserreserven der Erde, mit Schwerpunkt auf Schutz vor Verschmutzung und Erhalt von Wasserressourcen.
- **Atmosphäre**: die Gashülle der Erde, mit Schwerpunkt auf Reduzierung von Treibhausgasen, Schutz vor Luftverschmutzung und Anpassung an den Klimawandel.
- **Anthroposphäre**: der vom Menschen geschaffene Raum, mit Schwerpunkt auf nachhaltige Technologien, städtische Planung und ressourceneffiziente Produktion.
  - **Soziosphäre**: Soziale Strukturen, Gemeinschaften und zwischenmenschliche Beziehungen, mit Schwerpunkt auf soziale Gerechtigkeit, Bildung, Gesundheit und Armutsbekämpfung.
  - **Technosphäre**: Menschliche Technologien, mit Schwerpunkt auf nachhaltige Nutzung und Entwicklung, um negative Umweltauswirkungen zu minimieren und Effizienz zu fördern.
  - **Ökonomosphäre**: Wirtschaftliche Aktivitäten, Handel und Finanzsysteme, mit Schwerpunkt auf Ressourcen-effiziente, sozial gerechte und umweltfreundliche Wirtschaft.

- **Kultursphäre**: Kulturelle Aspekte wie Kunst, Sprache und Werte, mit Schwerpunkt auf kulturelle Vielfalt, den Schutz kulturellen Erbes und respektvollen Umgang mit verschiedenen Identitäten.

Diese Unterteilung ermöglicht es, die Auswirkungen menschlicher Aktivitäten auf die Umwelt zu kategorisieren und zielgerichtete Maßnahmen zur Nachhaltigkeit zu entwickeln. Es ist jedoch imperativ zu betonen, dass Nachhaltigkeit ein äußerst komplexer Sachverhalt ist, der interdisziplinär untersucht werden muss. Methoden, um komplexe Systeme zu verstehen, werden im nachfolgenden Abschnitt behandelt, das sich mit der Komplexitätsforschung befasst.

2.1.2 Komplexitätsforschung

Ökologische, ökonomische und soziale Fragen sind oft komplex und können nicht immer innerhalb der dafür prädestinierten Disziplinen beantwortet werden. Dies liegt daran, dass diese Fragen häufig durch die Wechselwirkung vieler unterschiedlicher Faktoren, die außerhalb der Disziplin liegen, mitbestimmt werden[17]. Die Komplexitätsforschung hat sich über die letzten Jahrzehnte als Forschungsgebiet etabliert, um allgemein komplexe interdisziplinäre Phänomene in einer disziplinierten Wissenschaft untersuchen zu können[29][11].

Komplexitätsforschung beschäftigt sich auf einer abstrakten Ebene mit vernetzten Strukturen, welche überall in der Natur und in der Gesellschaft zu finden sind. Beispiele sind Ökosysteme, soziale Netzwerke, Verkehrssysteme, Finanzmärkte oder die Evolutionstheorie als Ganzes[2][11]. In einem komplexen System sind die einzelnen Elemente miteinander verbunden. Diese Verbindungen können unterschiedliche Formen annehmen. Sie können gerichtet oder ungerichtet sein, hierarchisch oder flach. Die Art der Verbindungen hat einen großen Einfluss auf das Verhalten des Systems. Die Komplexitätsforschung untersucht die Wechselwirkungen zwischen den Elementen eines komplexen Systems. Sie kann dazu beitragen, diese Systeme besser zu verstehen und zu modellieren und ist daher auch für die Nachhaltigkeitsforschung von Bedeutung[40].

Stabilität eines Systems wird durch Rückkopplungseffekte erzielt, die sich selbst verstärken. Diese Rückkopplungseffekte können sowohl positiv als auch negativ sein. Positive Rückkopplungseffekte halten das System stabil. Negative Rückkopplungseffekte führen zu einer Destabilisierung des Systems und führt dazu, dass das System in einen neuen stabilen Zustand übergeht.

Beispielsweise erzeugt der Amazonas-Regenwald seinen eigenen Regen über den El Niño-Southern Oscillation (ENSO) Effekt[33]. Dies ist ein positiver Rückkopplungseffekt, der zu kontinuierlichem Regen führt und den Regenwald in einem stabilen Zustand hält. Bei fortschreitender Abholzung wird dieser Rückkopplungseffekt jedoch schwächer, bis er schließlich verschwindet[33]. Das System gerät dann in einen neuen Zustand, der durch einen anderen Rückkopplungseffekt bestimmt wird. Im Falle des Amazonas-Regenwalds könnte dies zu einer wüstenartigen Landschaft führen. Wenn die Rückkopplungseffekte umschlagen und das System in einen neuen Zustand übergeht, wurde ein Kipppunkt überschritten. Kipppunkte zu identifizieren und Auswirkungen abzuschätzen, spielen eine zentrale Rolle in der Nachhaltigkeitsforschung[1]. In dem 2015 veröffentlichten Paper „Planetary Boundaries: Guiding Human Development on a Changing Planet“[61] werden weitere Vektoren beschrieben, die katastrophale Folgen haben, wenn Kipppunkte, überschritten werden.

Eigenschaften stabiler Systeme, Rückkopplungseffekte, Kipppunkte und weiteres sind zentrale zu beantwortende Fragen in Nachhaltigkeitsdiskussionen. Die Komplexitätsforschung erlaubt somit die Betrachtung von Nachhaltigkeitsfragestellungen aus einem breiten, holistischen Blickwinkel.

2.1.3 Wirtschaftlicher Wandel

Wie eingangs des Kapitels erörtert, ist das seit dem 18., 19., 20. und nun im 21. Jahrhundert entwickelte und weiterentwickelte Wirtschaftssystem von zentraler Bedeutung, um die Ursachen, Folgen und Lösungen einer nachhaltigen Entwicklung zu begreifen. Das Wirtschaften, das ursprünglich über Tauschhandel und später hauptsächlich über den Handel von Waren gegen Währung durchgeführt wird, hatte lange Zeit keine gravierenden Auswirkungen auf die Umweltsphären. Dies änderte sich mit der Etablierung des Kapitalismus Ende des 18. Jahrhunderts. Zahlreiche Bücher[41][56][55][60][58][32], beschäftigen sich zu verschiedenen Zeiten und aus unterschiedlichen Blickwinkeln heraus, eingehend mit diesem Thema. Es lässt sich jedoch feststellen, dass unser Wirtschaftssystem zu einem hohen Wohlstand in der Gesellschaft geführt hat, was sich in einer höheren Lebenserwartung, gesteigerter Mobilität und erhöhtem Konsum manifestiert[3]. Allerdings muss das Wirtschaftssystem weiterentwickelt werden, um den Erkenntnissen der Nachhaltigkeitsforschung gerecht zu werden. Der 1972 veröffentlichte Bericht „Limits of Growth“[22] stellt unter anderem die folgenden zwei grundlegende Beobachtungen auf:

- Die Erde hat endliche natürliche Ressourcen, die durch steigende Bevölkerungszahlen, zunehmenden Konsum und fortschreitende Industrialisierung erschöpft werden.
- Exponentielles Wachstum führt Zeit nah zum Zusammenbruch der Umweltsphären.

Um den Umfang dieser Arbeit zielgerichtet zu wahren, wird dies nicht weiter ausgeführt und auf zuvor genannte Quellen verwiesen. In den nachfolgenden zwei Unterabschnitten werden zwei Konzepte beschrieben, die eine Rolle in der nachhaltigen Softwareentwicklung spielen, aber übergreifende Konzepte darstellen.

Rebound-Effekt

Der Rebound-Effekt beschreibt das Phänomen, dass nach einem Effizienzgewinn kein reduzierter, sondern ein gleichbleibender oder sogar erhöhter Verbrauch zu beobachten ist[38]. Dies kann in verschiedenen Kontexten auftreten, wie im Straßenverkehr durch den Ausbau einer Autobahn, der nicht zu weniger, sondern zu gleichbleibenden Staus führt, oder im Energieverbrauch, wie im nachfolgenden Diagramm 2.3 illustriert.

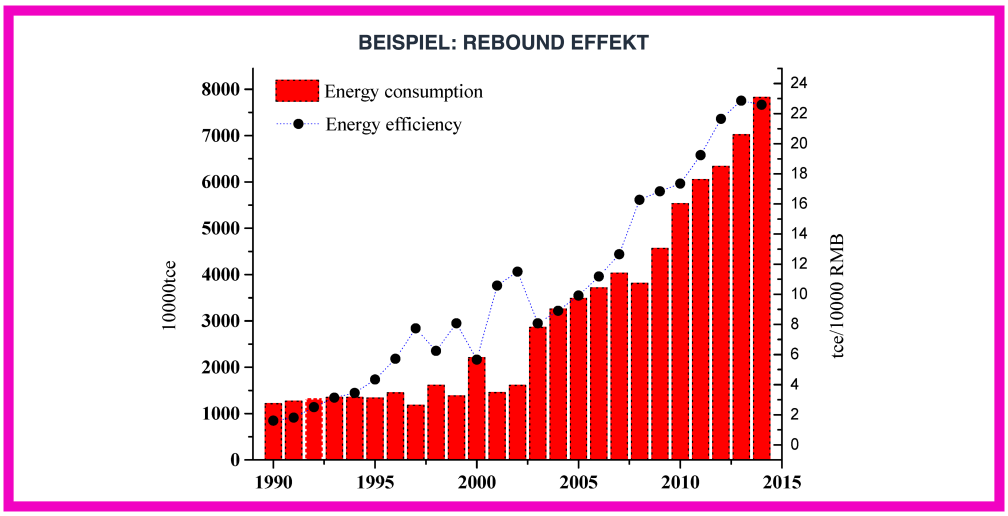

Abbildung 2.3: Beispiel für den Rebound-Effekt beim Energieverbrauch im Bausektor in China[23]

Das Diagramm zeigt, dass trotz steigender Energieeffizienz der Energieverbrauch weiter zunimmt. Der Rebound-Effekt ist keine naturgegebene Notwendigkeit, sondern eine Konsequenz unseres auf Ressourcennutzung ausgelegten Wirtschaftssystems.

Produktlebenszyklus

Nachhaltiges Handeln beinhaltet einen vorausschauenden, bewussten und ethischen Umgang mit Ressourcen. In der Produktentwicklung spiegelt sich dies in Konzepten wie dem nachhaltigen Produktlebenszyklus wider, der in dem nachfolgenden Diagramm 2.4 dargestellt ist[35].

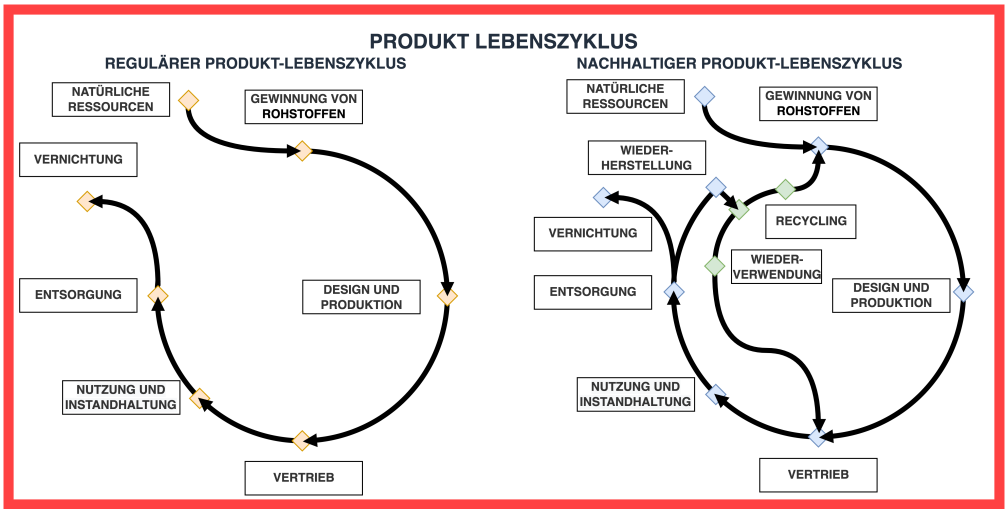

Abbildung 2.4: Gegenüberstellung adaptierter Lebenszyklusdiagramme[35]

Das linke Diagramm zeigt einen beispielhaften „regulären“ Lebenszyklus eines Produkts, der von Erstellung bis zur Entsorgung des Produktes gedacht ist. Das rechte Diagramm präsentiert eine Weiterentwicklung dieses Konzepts, bei der eine Weiter- oder Wiederverwendung der eingesetzten Ressourcen vorgesehen ist. Nachhaltige Produktlebenszyklen sind ein wesentlicher Bestandteil einer Kreislaufwirtschaft, welches als weiteres Konzept entwickelt wird, aber hier nicht weiter ausgeführt wird[25][9].

2.1.4 Klimawandel und die Kategorisierung von Emissionen

Der Klimawandel steht im Zentrum der globalen Nachhaltigkeitsbemühungen und ist eine der drängendsten Herausforderungen unserer Zeit. Die überwältigende wissenschaftliche Evidenz, dargestellt durch Institutionen wie den IPCC (Intergovernmental Panel on Climate Change), zeigt, dass die durch menschliche Aktivitäten verursachten Treibhausgasemissionen zu einer beispiellosen Erwärmung des Planeten führen. Diese Erwärmung hat weitreichende Auswirkungen auf natürliche und menschliche Systeme, von steigenden

Meeresspiegeln und zunehmenden Wetterextremen bis zu Auswirkungen auf die Nahrungsmittelsicherheit und globale Gesundheit.

In diesem Kontext ist es wichtig, die Treibhausgasemissionen detailliert zu erfassen und zu managen. Das Greenhouse Gas Protokoll (GHG Protocol)[70], was über NGOs in der Industrie entwickelt und flächendeckend Verwendung findet, bietet hierfür einen global anerkannten Rahmen[30]. Durch die Kategorisierung der Emissionen in Scope 1, Scope 2 und Scope 3 ermöglicht das GHG Protocol Organisationen, ihre direkten und indirekten Emissionen umfassend zu erfassen und Strategien für deren Reduzierung zu entwickeln.

- **Scope 1 - Direkte Emissionen**: Diese Kategorie umfasst alle direkten Emissionen, die aus Quellen stammen, die ein Unternehmen kontrolliert. Verbrennung von Kraftstoffen bei dem Betrieb von Maschinen, Emissionen aus Abfallprodukten oder im landwirtschaftlichen Betrieb wären Beispiele dafür.
- **Scope 2 - Indirekte Emissionen aus bezogener Energie**: Scope 2 bezieht sich auf Emissionen, die aus der Erzeugung von Strom, Dampf, Heizung oder Kühlung resultieren, die ein Unternehmen bezieht. Hier liegt der Fokus auf der Wahl der Energiequellen und der Förderung erneuerbarer Energien.
- **Scope 3 - Weitere indirekte Emissionen**: Diese Kategorie erfasst alle anderen indirekten Emissionen, die in der Wertschöpfungskette eines Unternehmens anfallen. Es umfasst Aspekte wie die Herstellung und den Transport von Zulieferprodukten sowie die Nutzung und Entsorgung der verkauften Produkte.

Das GHG Protokoll setzt einen wichtigen Rahmen für die Erfassung von Treibhausgasemissionen. Das Protokoll unterscheidet allerdings nicht zwischen linearen und zirkulären Wirtschaftsmodellen, was zu einem potenziellen Nachteil für Unternehmen führt, die kreislauforientierte Praktiken verfolgen[71]. Es gibt eine Reihe weiterer Modelle und Standards, die im weiteren Verlauf dieser Arbeit nicht relevant sind und daher nicht diskutiert werden.

2.1.5 Gesellschaftlich-politische Verständigungen

Nachhaltigkeitsherausforderungen sind, wie in vorherigen Abschnitten beschrieben, oft durch komplexe Zusammenhänge charakterisiert. Diese umfassen auch soziale Dimensionen, die durch gesellschaftlich-politische Vereinbarungen adressiert werden. Die Verein-

ten Nationen (United Nations, UN) haben über die Jahrzehnte hinweg eine signifikante Rolle in der Entwicklung und Umsetzung dieser gesellschaftlich-politischen Absprachen gespielt. Insbesondere auf der bereits angesprochenen (vgl. 2.1.1) UN-Brundtland-Konferenz im Jahre 1987 wurde der Begriff der nachhaltigen Entwicklung erstmals umfassend definiert[67].

Seit der Brundtland-Konferenz wurden zahlreiche weitere Konferenzen organisiert, wie die jährlichen Klimakonferenzen der Vereinten Nationen (COP), und Abkommen initiiert, die auf regionaler, nationaler und globaler Ebene angesetzt sind. Beispiele für solche Abkommen sind die UN Millenniums-Entwicklungsziele (2000–2015)[46], die UN Rio+20-Konferenz (2012)[47], das Pariser Klimaabkommen (2015)[48] oder der European Green Deal (2019)[16].

Im Folgenden werden zwei Konzepte aufgegriffen, die auch in der nachhaltigen Softwareentwicklung von Bedeutung sind, jedoch als übergreifende Konzepte verstanden werden. Erstens, das Konzept der Allmende (Commons) bezieht sich auf den Umgang mit gemeinschaftlich genutzten Ressourcen wie Wasser, Luft und Land und betont die gemeinsame Verantwortung für ihre nachhaltige Nutzung[53]. Zweitens, der ökologische Fußabdruck, ein Ansatz zur Approximation der von einer Person, Organisation oder Entität direkt oder indirekt eingesetzten Ressourcen. Im nächsten Unterkapitel wird die Schnittstelle zwischen Software und Nachhaltigkeit beleuchtet.

2.2 Schnittpunkte der Digitalisierung und der Nachhaltigkeitstransformation

In diesem Unterkapitel werden die Verbindungen zwischen Digitalisierung und Nachhaltigkeitstransformation beleuchtet. Es werden Themen wie nachhaltige Softwareentwicklung und der Energieverbrauch von Rechenzentren aufgeführt. Diese beiden Bereiche werden im weiteren Verlauf der Arbeit eingehender behandelt. Wie bereits zuvor wird auch in diesem Abschnitt die Vielfalt der Themen rund um Digitalisierung und Nachhaltigkeit trotz des begrenzten Umfangs deutlich gemacht. Dabei wird eine knappe Auflistung bevorzugt, ohne die Themen weiter zu vertiefen.

In einem 2001 veröffentlichten Bericht an die OECD wurden die Auswirkungen des ICT-Sektors (Informations- und Kommunikationstechnologie) auf die Nachhaltigkeitstransformation dargelegt[8]. Der Bericht unterscheidet dabei erstens zwischen Auswirkungen

erster, zweiter und dritter Ordnung, die als zeitliche Kategorien zu verstehen sind, und zweitens zwischen positiven und negativen Auswirkungen. In der nachstehenden Tabelle 2.1 finden sich hierzu beispielhafte Aufschlüsselungen. Zu den positiven Auswirkungen gehören etwa die Unterstützung, die Software bei der Umweltüberwachung leistet, sowie die Immaterialisierung von Produkten und Dienstleistungen. Negative Auswirkungen umfassen unter anderem die ökologischen Folgen, die mit der Produktion von ICT-Geräten einhergehen, sowie den Rebound-Effekt (siehe Abschnitt 2.1.3).

	Positive Wirkungen	**Negative Wirkungen**
Auswirkungen erster Ordnung	ICT-Anwendungen im Umweltbereich *z.B. Umweltüberwachung*	Umweltauswirkungen der Produktion & Nutzung von ICTs *z.B. Elektronikschrott*
Auswirkungen zweiter Ordnung	Immaterialisierung Strukturwandel *z.B. digitale Verzeichnisse*	Unvollständige Substitution *z.B. 'white vans' zusätzlich zu den privaten Einkaufstouren*
Auswirkungen dritter Ordnung	Änderungen des Lebensstils *z.B. Grüner Konsum*	rebound effect *z.B. Zunahme des Reiseverkehrs*

Tabelle 2.1: ICT Auswirkungen auf die Nachhaltigkeit[8]

Weitere Arbeiten, die die Schnittstellen thematisieren, werden in [5] und [64] erwähnt. Die Schnittstellen zwischen Digitalisierung und Nachhaltigkeitstransformation werden in den nachfolgenden Abschnitten entsprechend den im 2.1.1 dargestellten Aspekten der Nachhaltigkeit strukturiert. Die identifizierten Schnittpunkte sind im Diagramm 2.5 visuell aufbereitet. Diese Auflistung beansprucht keine Vollständigkeit und resultiert nicht aus einer strukturierten Literaturanalyse. Sie dient, ähnlich wie das vorherige Kapitel, dazu, das breite Spektrum der Verbindung zwischen Software und Nachhaltigkeit zu verdeutlichen. Eine Bewertung der Schnittpunkte wird nicht vorgenommen.

2.2.1 Ökologische Gesichtspunkte

- **Abbau seltener Erden**: Die Produktion elektronischer Geräte ist stark abhängig von seltenen Erden. Der Abbau, die Verhüttung und die Raffination dieser Erze gehen jedoch häufig mit bedeutenden Umweltbelastungen einher. Dazu gehören die Zerstörung natürlicher Landschaften, die oft in wüstenähnliche Gebiete verwandelt

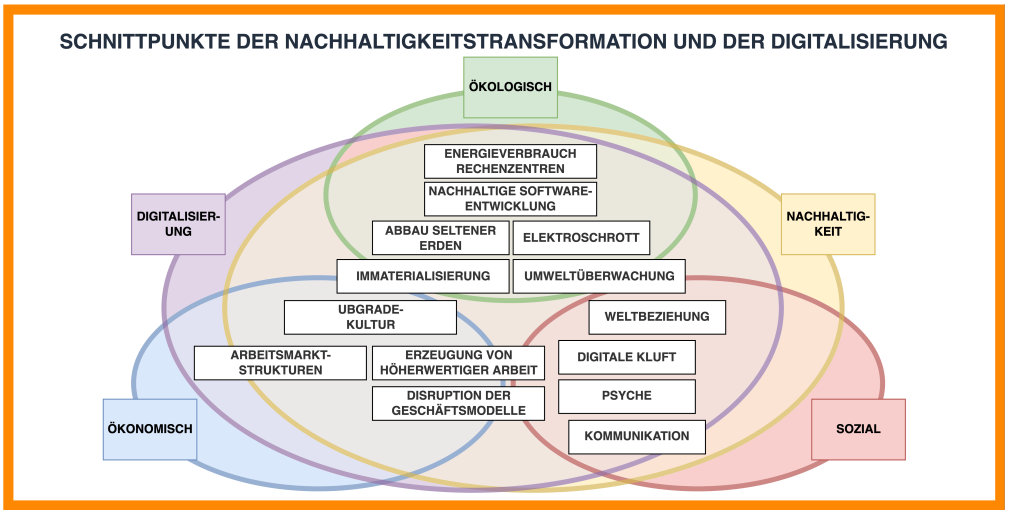

Abbildung 2.5: Schnittpunkte der Nachhaltigkeitstransformation und der Digitalisierung nach Nachhaltigkeitsaspekten

werden, die Verschmutzung von Wasserressourcen sowie die Entstehung schädlicher Nebenprodukte[66].

- **Elektroschrott**: Das zunehmende Aufkommen von Elektroschrott, das nicht mit der Recyclingrate Schritt hält, stellt ein wachsendes Problem dar[49].
- **Umweltüberwachung**: Digitalisierung ermöglicht verbesserte Umweltüberwachung und Datenanalyse, was zu effektiveren Schutzmaßnahmen führen kann[43]. Initiativen wie die Global Earth Observation System of System (GEOSS), UN Global Environment Monitoring System (GEMS) oder NASA Earth Observing System (EOS) sind Beispiele dafür.
- **Immaterialisierung**: Die Digitalisierung führt zur Immaterialisierung vieler Produkte und Dienstleistungen (wie E-Books oder Streaming-Dienste), was den physischen Ressourcenverbrauch reduzieren kann[64].
- **Energieverbrauch von Rechenzentren und ICT Infrastruktur**: Der wachsende Energiebedarf von Rechenzentren ist eine bedeutende ökologische Herausforderung[42]. Dieser Bereich wird im weiteren Verlauf der Arbeit vertieft.
- **Green IT und nachhaltige Softwareentwicklung**: Initiativen zur Reduzierung der Umweltauswirkungen von IT-Systemen, einschließlich energieeffizienter Hard-

ware und Software. Dieser Bereich wird im weiteren Verlauf der Arbeit in Bezug auf nachhaltige Cloud Native Infrastrukturen vertieft (siehe 3).

2.2.2 Ökonomische Gesichtspunkte

- **Automatisierung von Prozessen und Erzeugung von höherwertiger (erfüllender) Arbeit**: Durch den Einsatz von digitalen Technologien werden repetitive und manuelle Aufgaben verringert, was zu einer Verschiebung hin zu anspruchsvolleren und kreativeren Tätigkeiten führt. Diese Entwicklung birgt das Potenzial, die Arbeitszufriedenheit zu erhöhen und neue Berufsfelder zu schaffen[27].
- **Upgrade-Kultur**: In der Gesellschaft führt die ständige Aktualisierung von Produkten und Dienstleistungen zu einer „Upgrade-Kultur“. Diese begünstigt einerseits Innovationen, führt andererseits jedoch auch zu schnellerer Obsoleszenz und erhöhtem Ressourcenverbrauch.[44].
- **Disruption etablierter Geschäftsmodelle**: Digitale Plattformen wie Uber, Airbnb und Facebook haben traditionelle Geschäftsmodelle in Bereichen wie Transport, Gastgewerbe und Medien grundlegend verändert. Diese Disruptionen bieten Chancen für Innovation und Effizienzsteigerung, stellen aber auch Herausforderungen für bestehende Unternehmen und Regulierungsrahmen dar[54][64][27][4][57].
- **Veränderung der Arbeitsmarktstrukturen**: Digitalisierung führt zur Entstehung neuer Berufsfelder[27].

2.2.3 Soziale Gesichtspunkte

- **Weltbeziehung**: „Durch technischen Fortschritt ist es mittlerweile möglich, über den Bildschirm alle möglichen Aufgaben erledigen zu lassen. Über den Bildschirm treten wir in Kontakt mit der Welt, was in endgültiger Konsequenz zur Entfremdung und allgemein einem *ärmeren* Leben führt. Dies wirft [. . .] interessante Frage auf, wie sich die Natur des menschlichen und seines biografischen Weltverhältnisses insgesamt ändert, wenn Bildschirm zum Leitmedium nahezu aller Weltbeziehungen werden“.[59]
- **Globale Vernetzung und Kommunikation**: Die Art und Weise, wie wir kommunizieren, hat sich durch digitale Plattformen grundlegend gewandelt[13].

- **Digitale Kluft und soziale Ungleichheit**: Die digitale Kluft, also die ungleiche Verfügbarkeit und Nutzung digitaler Technologien, verstärkt bestehende soziale Ungleichheiten[52].
- **Einfluss auf psychische Gesundheit**: Digitale Medien, insbesondere soziale Netzwerke beeinflussen die psychische Gesundheit von Nutzern[65].
- **Digitale Unabhängigkeit**: Digitale Unabhängigkeit bezieht sich auf die Fähigkeit, die digitale Umgebung und Daten zu kontrollieren. Dies ist wesentlich für die Wahrung von Datenschutz und Datensouveränität und unterstützt die Wahlmöglichkeit zwischen verschiedenen digitalen Diensten. Die digitale Unabhängigkeit ist beispielhaft nach [24] im Kontext der EU von Bedeutung, da sie die Autonomie und Widerstandsfähigkeit gegenüber externen digitalen Mächten stärkt. Offene Standards und digitale Bildung sind dabei Schlüsselelemente, um eine nachhaltige digitale Zukunft zu sichern.

3 Nachhaltige Softwareentwicklung

In diesem Kapitel wird das Thema der nachhaltigen Softwareentwicklung Schritt für Schritt ergründet. Im ersten Unterkapitel werden Forschungsarbeiten zur grünen Softwareentwicklung beschrieben. Im zweiten Unterkapitel wird der Blick auf die Cloud und im speziellen Cloud Native gelenkt und Cloud Native Sustainability als Bereich eingeführt. Im dritten und letzten Unterkapitel werden die Abstraktionsschichten eines Softwaresystems durchleuchtet und beschrieben, welche Verantwortung jede Abstraktionsschicht trägt, um Ressourcen effektiv einzusetzen und einen Überblick über den Nachhaltigkeitsfußabdruck eines Systems zu erhalten.

3.1 Forschungsgebiet: Grüne Softwareentwicklung

Während der 1980er- und 1990er-Jahre wurden in der Informatik Schlüsselinnovationen wie GUIs (wie Windows), Programmiersprachen (wie C und C++), Betriebssysteme (wie Unix), Protokolle (wie TCP/IP) und das Internet (WWW) entwickelt. Parallel dazu wurde in allgemeinen wissenschaftlichen und politischen Kreisen das Thema Nachhaltigkeit erstmalig breit diskutiert (vgl. Abschnitt 2.1.5). Die Nachhaltigkeits-Diskussionen kamen erst einige Zeit später in die Informatik an. Es kann die These formuliert werden, dass diese Entwicklung Hand in Hand mit dem steigenden Einfluss der Informatik auf Gesellschaft und Wirtschaft ging.

In dem 1995 erschienen Paper „A Plea for Lean Software“[69] wurde darauf hingewiesen und kritisiert, dass Software über die Jahrzehnte stetig mehr Ressourcen (CPU, Speicher) benötigt, dies aber nicht im Verhältnisse zu der Weiterentwicklung von Software steht. In dem Paper wurde geäußert, dass „vor etwa 25 Jahren ein [...] Texteditor mit nur 8.000 Byte Speicherplatz entwickelt werden konnte. (Moderne Programmeditoren verlangen das 100-fache!) [...] Ist all diese aufgeblähte Software schneller geworden? Ganz im Gegenteil. Wäre die Hardware nicht tausendmal schneller, wäre moderne Software gänzlich unbrauchbar.“[69]. Als Gründe werden inflationäres Feature Management, Zeitdruck

und Komplexität der zu bewältigenden Aufgaben hervorgebracht. Diese Beobachtung ist semantisch kongruent mit der Beobachtung von Hans Carl von Carlowitz aus dem 18. Jahrhundert (vgl. Abschnitt 2.1).

In dem 2015 erschienen Paper „Sustainability Design and Software: The Karlskrona Manifesto“[6] wird nachhaltige Softwareentwicklung als Bereich in der Informatik vorgeschlagen. Ein Fokus liegt auf den Aspekten der Nachhaltigkeit, die in Abschnitt 2.1.1 beschrieben wurden. Nachhaltigkeit wird als Disziplin-übergreifend beschrieben.

Auf einer Meta-Ebene kann festgestellt werden, dass viele Forschungsarbeiten einen indirekten Einfluss auf die Nachhaltigkeit haben. Beispielsweise betrifft dies die Entwicklung leistungsfähigerer Algorithmen und Protokolle, die Anforderungsentwicklung sowie die Erarbeitung neuer Architekturmuster für den geschickten Aufbau zukunftssicherer, erweiterbarer und wartbarer Software. Diese Bereiche sind nicht nur für den Fortschritt des jeweiligen Fachgebiets von Bedeutung, sondern tragen auch einen wesentlichen Wert zur Nachhaltigkeit bei. Wie im vorherigen Kapitel beschrieben, findet Nachhaltigkeit in vielfältigen Kontexten Bedeutung. In der Informatik wurden daher Forschungsarbeiten, die sich speziell mit ökologischer Nachhaltigkeit befassen, als grüne Softwareentwicklung (Green Software Engineering) beschrieben, um den Schwerpunkt klarzumachen. In diesem Abschnitt der Arbeit konzentrieren wir uns daher hauptsächlich auf die grüne Softwareentwicklung.

Um dieses Unterkapitel zielgerichtet und nicht zu langatmig zu gestalten, werden einige Themen vorab ausgeschlossen. Arbeiten zur IT-Hardware werden hier nicht betrachtet, finden jedoch im nachfolgenden Abschnitt 3.3.1 kurz Beachtung. Ebenfalls ausgeschlossen wird der Einsatz von Software zur Messung von Nachhaltigkeitsaspekten, wie der Einsatz von Software für Wettervorhersagen, ein Bereich, der im vorherigen Abschnitt 2.2.1 aufgezeigt wurde.

Dieses Unterkapitel klassifiziert nachhaltige Softwareentwicklung zunächst als einen spezifischen Bereich innerhalb der ethischen Softwareentwicklung. Es untersucht die Einstellung von Softwareentwicklern zur Nachhaltigkeit und erläutert, wie Anforderungen als effektives Instrument dienen können, um Nachhaltigkeit unabhängig von spezifischen Prozessen strukturell zu integrieren. Anschließend wird das GREENSOFT-Modell vorgestellt, ein umfassendes Rahmenwerk, das den Lebenszyklus von Softwareprodukten mit Verfahrensmodellen, Nachhaltigkeitsmetriken und Werkzeugen verknüpft. Im folgenden Abschnitt werden Qualitätsstandards für Software und deren Erweiterung um Nachhaltigkeitsaspekte thematisiert. Der Schwerpunkt verlagert sich dann auf Prozesse, die zur

Förderung der Softwareentwicklung etabliert wurden, und wie diese um Aspekte der Nachhaltigkeit erweitert werden können. Abschließend wird die grüne Softwarearchitektur beleuchtet und es werden spezifische Domänen innerhalb der Informatik diskutiert, die mit grüner Software in Verbindung stehen.

3.1.1 Ethische Softwareentwicklung

Nachhaltige Softwareentwicklung kann als Unterbereich der ethischen Softwareentwicklung aufgefasst werden. Eine »Digitale Ethik« hat zum Ziel, dem Menschen dabei zu helfen, in der sich weiter modernisierenden Gesellschaft mit ihren vielfältigen digitalen Geräten und Anwendungen einen angemessenen Umgang mit diesen Technologien und ihren Auswirkungen zu finden[31]. Als Erweiterung der Ethik analysiert die »Digitale Ethik« [...], welche legitimen Handlungsoptionen sich aus der Entwicklung, dem Einsatz und der Anwendung digitaler Technologien ergeben[31].

Dies korrespondiert mit Forschungen im Feld der nachhaltigen Softwareentwicklung, welche essenziell hervorheben, dass wir als Entwickler von Softwaretechnologien für die Langzeitfolgen unserer Designs verantwortlich zeichnen[6]. Nach [36] wird der „Unsichtbarkeitsfaktor“ der ethischen Software genannt, der auch in Diskussionen über nachhaltige Softwareentwicklung von Bedeutung ist[45]. Der Unsichtbarkeitsfaktor beschreibt, dass die verborgenen internen Prozesse von Computersystemen unbeabsichtigte oder unethische Nutzungen fördern können, was sowohl eine Politik- als auch Konzeptlücke im korrekten Einsatz der Computertechnologie verursacht[36]. Nutzer sind sich der zugrundeliegenden Prozesse genutzten IT Ressourcen nicht umfassend bewusst, was die Identifizierung der tatsächlich verwendeten natürlichen Ressourcen erschwert.

3.1.2 Mindset und Grüne Software Anforderungen

In den Arbeiten [5], [34], [62] und [14] wird untersucht, was unter Nachhaltigkeit in der Softwareentwicklung verstanden wird und wie Anforderungen angepasst werden können, um Nachhaltigkeit im Softwareentwicklungsprozess zu berücksichtigen. Die Arbeit zeigt, dass die Komplexität und der Umfang des Themas unter Softwareentwicklern oft nicht erkannt wird und nur einige Bereiche bekannt sind. „In der Regel betrachteten die Befragten Nachhaltigkeit als eine Frage der Verfügbarkeit natürlicher Ressourcen und der Abfallreduzierung“[14]. In den Papern werden eine Reihe an Punkten besprochen, an den

Aufgabe	**Aktuelle Praxis**	**Schwerpunkt der künftigen Praxis**
Mindset	Die Welt ist ein Puzzle, und wir sollten die Probleme lösen.	Die Welt ist komplex, und wir sollten zuerst die Dilemmas verstehen.
Risikoerkennung	Konzentration auf die Funktionen und sofortigen Effekte, die die Stakeholder wünschen.	Unterstützung des Stakeholders, dass Systems im Ganzen zu verstehen. Einschließlich langfristige strukturelle Effekte und Risiken.
Risiko-identifikation	Identifizieren von Risiken, die eine fristgerechte Fertigstellung des Projekts innerhalb des Budgets bedrohen.	Berücksichtigung des breiteren Umfelds des Systems. Einbeziehung von strukturellen Effekten sowie Risiken, die sich über die Zeit entwickeln können.
Anforderungs-dokumentation	Aktuelle Vorlagen ignorieren langfristige Effekte und Nachhaltigkeitsüberlegungen	Vorlagen erfordern Informationen über Nachhaltigkeit als Designanliegen und unterstützen Analysten mit Checklisten
...	...	...

Tabelle 3.1: Softwareentwicklung – Praktiken für Nachhaltigkeit[5]

angesetzt werden muss, damit Nachhaltigkeit unter Softwareentwicklern besser verstanden und damit zielgerichteter diskutiert werden kann. In der Tabelle 3.1 ist dargestellt, wie dieser Gedankenwandel (oder Mindset) aussehen könnte.

3.1.3 GREENSOFT Model

Das Paper „The GREENSOFT Model: A reference model for green and sustainable software and it's engineering“[50] erarbeitet ein umfassendes Modell, das Grüne Softwareentwicklung beschreibt und Vorschläge macht, wie es integriert werden kann. Das Modell umfasst verbindet den Softwareproduktlebenszyklus mit Metriken, Verfahrensmodellen und Tools, die zusammenarbeiten, um Entwicklern von Software bei der Entwicklung, Wartung und Nutzung von Software in einer nachhaltigeren Weise ein Framework zu geben. Die genannten Bereiche sind in der Abbildung 3.1 dargestellt.

- **Lebenszyklus von Softwareprodukten**: Betont die Bedeutung der Berücksichtigung des gesamten Lebenszyklus von Softwareprodukten, von der Entwicklung über die Nutzung bis zur Entsorgung, um Nachhaltigkeit zu fördern.

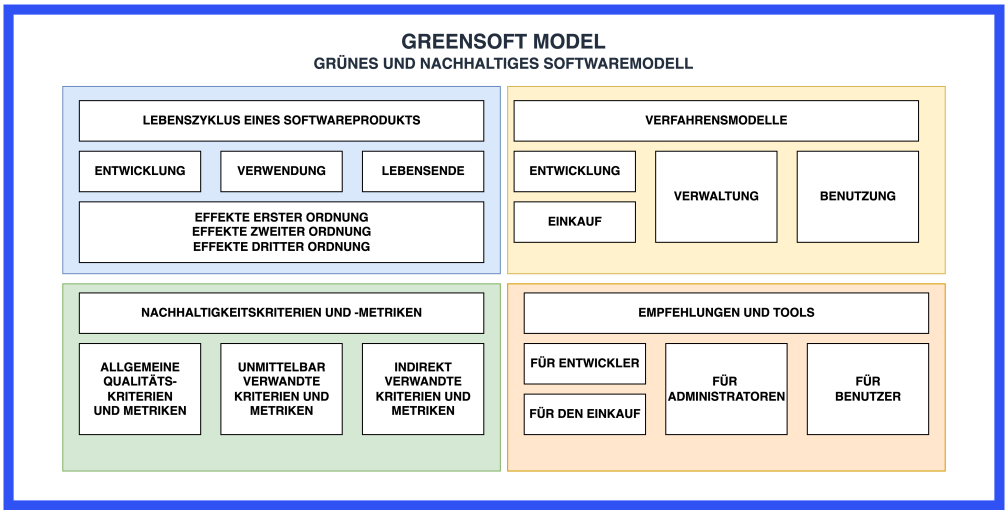

Abbildung 3.1: Das GREENSOFT Model ein Referenzmodell für „Grüne und nachhaltige Software“[50]

- **Nachhaltigkeitskriterien und -metriken**: Bietet Metriken und Kriterien für die Bewertung der Nachhaltigkeit von Softwareprodukten, um Entwickler und Unternehmen bei der Messung und Verbesserung ihrer Nachhaltigkeitsleistung zu unterstützen.
- **Verfahrensmodelle**: Stellt spezifische Verfahrensmodelle und Praktiken vor, die in verschiedenen Phasen der Softwareentwicklung und -wartung angewendet werden können, um die Nachhaltigkeit zu verbessern.
- **Empfehlungen und Tools**: Schlägt Werkzeuge und Best-Practices vor, die Stakeholdern helfen, Nachhaltigkeitsprinzipien in ihre Arbeit zu integrieren.

Das GREENSOFT-Modell zielt darauf ab, sowohl die direkten als auch die indirekten Auswirkungen der Softwareentwicklung und -nutzung auf die Nachhaltigkeit zu adressieren. Es bietet praktische Strategien für die Integration von Nachhaltigkeitsaspekten in den Softwareentwicklungsprozess. Die Einzelheiten des Modells werden hier nicht vertieft. In dem Modell nehmen Metriken eine zentrale Rolle ein, um präzise Prognosen zu ermöglichen, die für eine zukunftsorientierte nachhaltige Softwareentwicklung unerlässlich sind. Zudem wird das Lebenszyklusmodell beschrieben, das in Abschnitt 2.1.3 konzeptionell erörtert wurde. In den nachfolgenden Abschnitten 3.1.4 „Grüne Qualitätskriterien“ und 3.1.5 „Grüne Software-Entwicklungsmodelle“ werden Aspekte des GREENSOFT Modells weiter ausgeführt.

3.1.4 Grüne Qualitätskriterien

In [51] wird ein Qualitätsmodell für nachhaltige Softwareentwicklung vorgestellt, das in Abbildung 3.2 visualisiert ist.

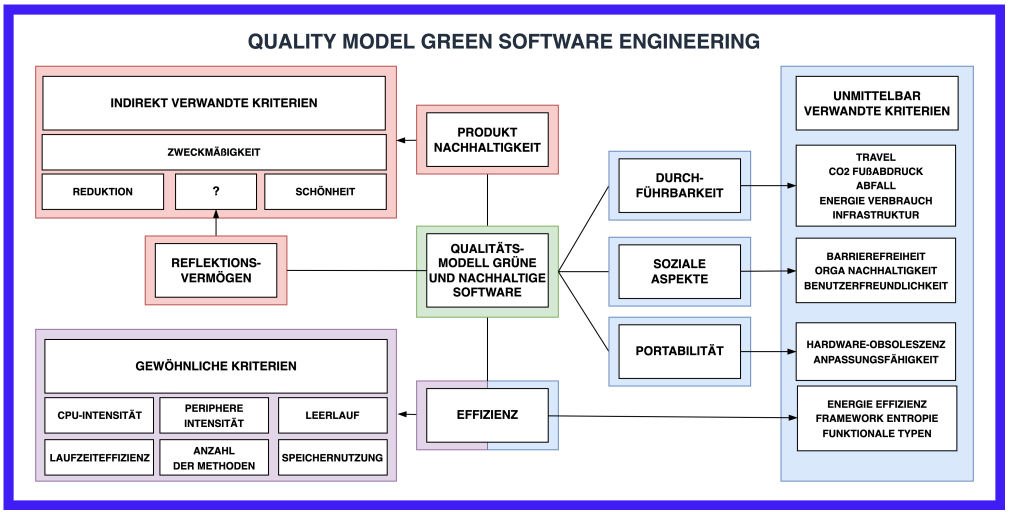

Abbildung 3.2: Qualitätsmodell für nachhaltige Software[50]

Gewöhnliche Kriterien (Common Quality Criteria): Die allgemeinen Qualitätskriterien basieren auf den bekannten und standardisierten Qualitätsaspekten für Software, die von der internationalen Organisation für Normung (ISO) herausgegeben wurden (vgl. 3.1.4).

Unmittelbar verwandte Kriterien (*Directly Related Criteria*): Im Gegensatz zu den gewöhnlichen Effizienzkriterien, die als Indikatoren für die Energieeffizienz betrachtet werden können, kann die Energieeffizienz eines Systems selbst in einem Referenzsystem eingestuft werden, mit dem Ziel, den Energieverbrauch in Bezug auf gelieferte Dienstleistungen zu optimieren. Hardware-Obsoleszenz betrachtet die Menge an Hardware, die ersetzt werden muss, bevor die nachhaltig lohnenswerte Lebensdauer erreicht ist. Adaptabilität und Portabilität sind ebenfalls relevant, ebenso wie die Machbarkeit, die die Auswirkungen der Produktentwicklung auf die nachhaltige Entwicklung bewertet. Dies umfasst Aspekte wie Reiseaufkommen, CO_2-Fußabdruck, Energieverbrauch und Ressourcenverschwendung. Des Weiteren werden soziale Aspekte der nachhaltigen Entwicklung wie Zugänglichkeit, Benutzerfreundlichkeit und organisationale Nachhaltigkeit in Betracht gezogen.

Indirekt verwandte Kriterien (*Indirectly Related Criteria*): Diese Kriterien umfassen den Einfluss der Software auf andere Branchen oder Anwendungsbereiche, indem sie Ressourcen und Energieeinsparungen ermöglichen. Diese könnten auch als transitive Kriterien benannt werden. Produktnachhaltigkeit fasst die Auswirkungen der Software auf andere Produkte und Dienstleistungen zusammen, einschließlich ihrer Anpassungsfähigkeit und systemischen Effekte. Dazu gehören Aspekte wie Zweckmäßigkeit, Reduktion und Schönheit, die den Beitrag der Software zur nachhaltigen Entwicklung bewerten.

3.1.5 Grüne Software-Entwicklungsmodelle

Dieser Abschnitt beleuchtet Verfahren der Softwareentwicklung, die darauf abzielen, Software nachhaltiger zu gestalten. Dabei werden der nachhaltige Software-Lebenszyklus, die Einbindung von Nachhaltigkeitsprozessen in die Softwareentwicklung gemäß ISO/IEC 12207 sowie die Methode des Green Agile erörtert.

Nachhaltiger Software Lebenszyklus

Zuvor wurde in Abschnitt 2.1.3 die Idee der nachhaltige Lebenszyklus Modells beschrieben. Dieser Ansatz findet auch in der Softwareentwicklung Beachtung, wie in Abbildung 3.3 dargestellt ist[50].

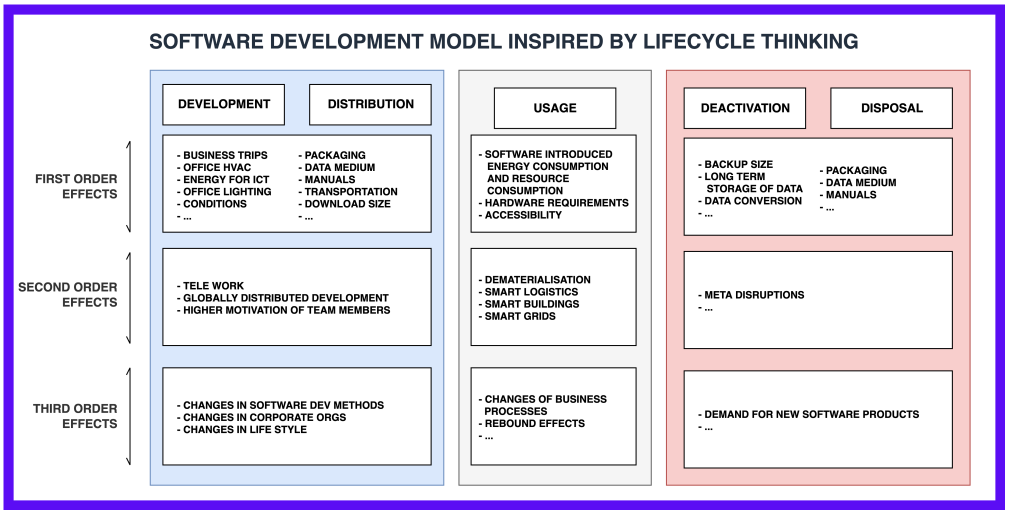

Abbildung 3.3: Software Development Prozess inspiriert bei von dem Lebenszyklus-Diagramm siehe Abschnitt 2.1.3[50]

In diesem Diagramm sind die Lebenszyklusphasen Entwicklung, Distribution, Nutzung, Deaktivierung und Entsorgung dargestellt. Darunter werden Kriterien aufgeführt, die in diesen Lebensphasen der Software unter dem Aspekt der Nachhaltigkeit relevant sind. Diese Kriterien sind in Erst-, Zweit- und Drittordnungskriterien sortiert, ein Schema, das auch zuvor verwendet wurde[8] (vgl. 2.2).

Nachhaltigkeitsprozesse in der Softwareentwicklung nach ISO/IEC 12207

ISO/IEC/IEEE 12207 ist eine Norm für Software-Lebenszyklus-Prozesse, die einen Rahmen für die Entwicklung, den Erwerb, die Bereitstellung und die Wartung von Softwareprodukten bietet. In [28] wird die Nachhaltigkeit in der Softwareentwicklung aus einer prozessorientierten Perspektive beleuchtet. Dies wird als Ergänzung zu der ISO/IEC 12207 vorgeschlagen. Es werden drei Schlüsselprozesse identifiziert: den nachhaltigen Managementprozess, den nachhaltigen Entwicklungsprozess und den nachhaltigen Qualifizierungsprozess. Diese Prozesse zielen darauf ab, sowohl Softwareprodukte als auch Softwareprozesse kontinuierlich in Richtung Nachhaltigkeit zu verbessern und den ökologischen Fußabdruck zu minimieren.

- **Nachhaltiger Managementprozess**: Der nachhaltige Managementprozess beinhaltet Phasen wie die vorläufige Festlegung von Nachhaltigkeitsprinzipien, Planung, Überwachung und Kontrolle der Lieferanten-Nachhaltigkeit.
- **Nachhaltiger Entwicklungsprozess**: Der nachhaltige Entwicklungsprozess fokussiert auf die Integration von Nachhaltigkeitstechniken und -methoden in den Softwareentwicklungsprozess
- **Nachhaltiger Qualifizierungsprozess**: Der nachhaltige Qualifizierungsprozess befasst sich mit der Nachhaltigkeit externer Ressourcen

Green Agile

Agile Softwareentwicklung ist mehr eine Mentalität, dass Flexibilität, Teamarbeit, Kundenorientierung und die Bereitschaft zur Anpassung an Veränderungen betont, als eine konkrete Methode zur Softwareentwicklung. Methoden wie Scrum bringen systematisch Agilität in den Entwicklungsprozess. Neben Scrum gibt es auch andere agile Methoden wie Kanban, Extreme Programming (XP) und Feature-Driven Development (FDD), die

auf kurze Feedbackschleifen, Iterationen und die inkrementelle Entwicklung von Softwarefeatures setzen.

In dem Artikel [21] wird ein agiler Softwareentwicklungsprozess vorgeschlagen, der aufzeigt, wie Nachhaltigkeit integriert werden kann. Im Mittelpunkt steht ein Journal, das die Nachhaltigkeitsaspekte des Softwareprojekts dokumentiert. Dieses Journal wird in regelmäßigen Abständen präsentiert und zum Zeitpunkt der Softwareveröffentlichung als Bericht aufbereitet. Die zugrundeliegende Idee lässt sich auf agile Entwicklungsmethoden anwenden. Im Artikel wird dies am Beispiel von Scrum demonstriert, was in der Abbildung 3.4 visualisiert wird.

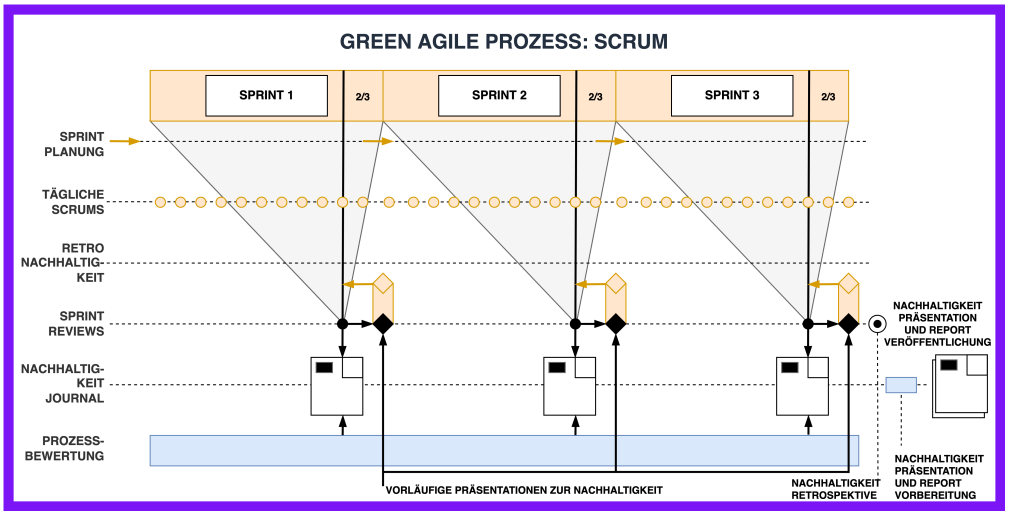

Abbildung 3.4: Green Agile Prozess in SCRUM [21]

3.1.6 Grüne Software-Architektur

Die Softwarearchitektur widmet sich der strukturellen Entwicklung von Software, um deren Qualität zu sichern. Sie empfiehlt den Einsatz spezifischer Architekturmuster und Entwurfsmodelle, die nicht nur die Nachhaltigkeit der Software durch zukunftsfähige Lösungsansätze fördern, sondern auch pragmatische Wege zur Problemlösung bieten. Diese Muster und Modelle tragen dazu bei, Software zu entwickeln, die langfristig nutzbar und wartbar bleibt. Im Fokus dieses Abschnitts steht, wie eingangs erwähnt, die ökologische Nachhaltigkeit von Software, die in den folgenden Unterabschnitten zu Verfahrensmodellen von Softwarearchitektur detaillierter erörtert wird. Es wird nicht auf alle

Softwarearchitektur-Modelle und -Muster eingegangen, sondern auf jene, die in der Recherche qualitativ hochwertige Arbeiten zur grünen Softwareentwicklung hervorgebracht haben.

Grüne Service Oriented Architecture (SOA)

Service Oriented Architecture (SOA) ist ein Architekturstil in der Softwareentwicklung, der sich durch die Nutzung diskreter Services von einem monolithischen Design abgrenzt. Diese Services sind über ein Netzwerk zugänglich, unabhängig voneinander einsetzbar und erlauben es, wiederholbare Geschäftsprozesse flexibel und effizient zu gestalten. SOA fördert die Wiederverwendbarkeit und Modularität durch unabhängige, über Netzwerke kommunizierende Komponenten, die eine robuste Grundlage für die Nutzung und Integration verschiedener Dienste bieten, unabhängig von Anbietern, Produkten und Technologien[10].

In dem Paper [39] werden die Problembereiche grüner Software im Kontext der SOA untersucht. Das Problem wird in die Aspekte Prozessbewusstsein (*Process Awareness*), Servicebewusstsein (*Service Awareness*) und Entwicklerbewusstsein (*People Awareness*) unterteilt. Diese Bereiche beziehen sich auf die direkten und indirekten Umweltauswirkungen der Ausführung und Entwicklung von Software-Services sowie auf die Nutzung von Services zur Überwachung und Messung dieser Auswirkungen, um das Bewusstsein für ökologische Fragen zu schärfen. In der Abbildung 3.5 ist dies visualisiert.

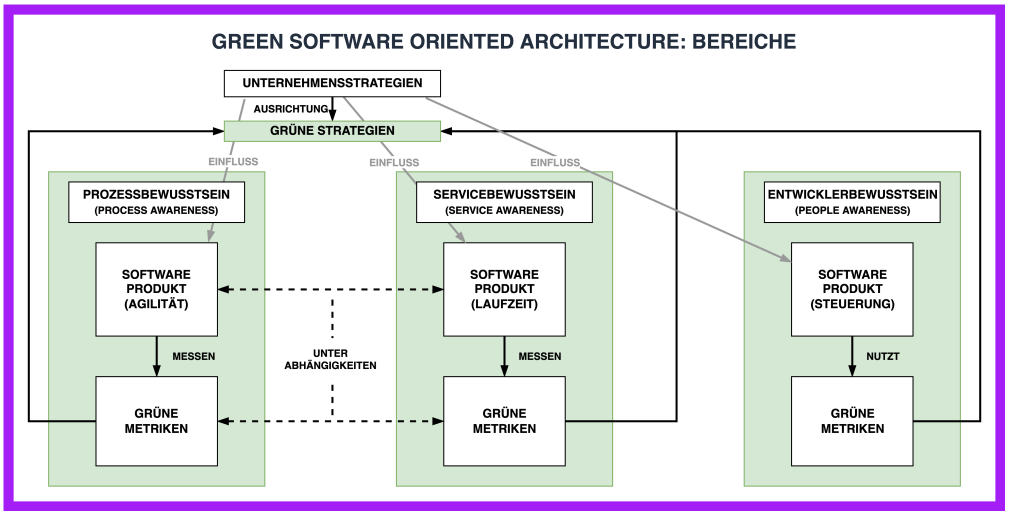

Abbildung 3.5: Grüne Problembereiche in der Serviceorientierung[39]

Als Lösungsvorschlag wird das Konzept des „Service Greenery“ als Ansatz zur Neugestaltung umweltbewusster Service-basierter Anwendungen (SBAs) vorgestellt, mit dem Ziel, den CO_2-Fußabdruck zu minimieren und die Nachhaltigkeit durch die Integration von Umweltstrategien als Service und durch die Bereitstellung von grünen Metriken zur Messung der Umweltauswirkungen zu erhöhen. In der Abbildung 3.6 dies visualisiert.

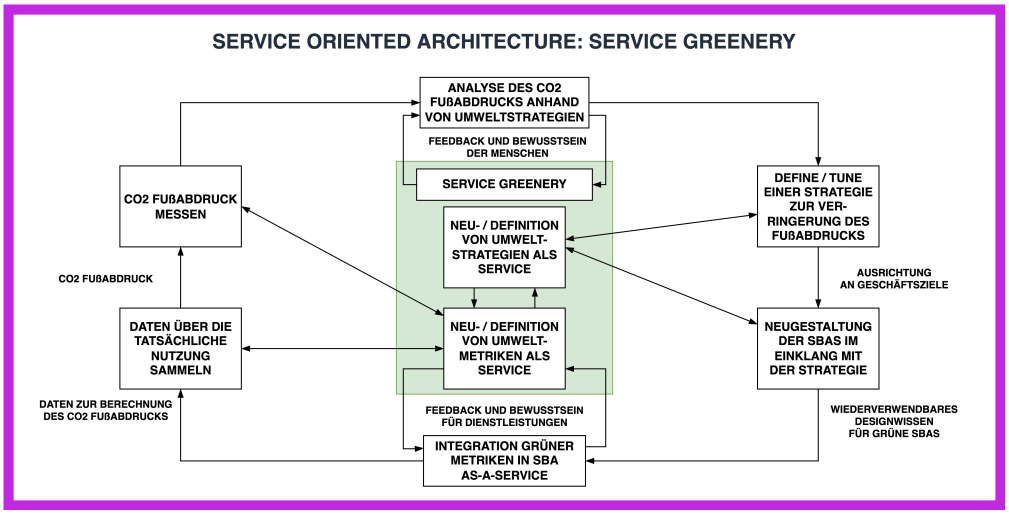

Abbildung 3.6: Service Oriented Architecture – *Service Greenery* als zentrale Rolle bei der Neugestaltung umweltbewusster Service Based Applications (SBAs)[39]

Diese Ansätze und Metriken sind entscheidend, um die Umweltauswirkungen von Software-Services zu messen und zu optimieren, was letztlich zu einer grüneren und nachhaltigeren Gestaltung von Service-orientierten Architekturen führt.

Diskussion: Pragmatische Ansätze grüner Softwarearchitektur

Das „Green Architecture Framework“[15], vorgestellt in der Arbeit von Henrik Bærbak Christensen, bietet eine pragmatische Strategie zur Integration von Nachhaltigkeit in der Softwareentwicklung. Durch die Kombination verschiedener Taktiken und Prozesse, die flexibel an das jeweilige System angepasst werden können, zielt das Framework darauf ab, Entwicklern konkrete Ansatzpunkte für energieeffizienteres Design zu geben. In der Abbildung 3.7 sind die Taktiken und Prozesse dargestellt.

Die im Framework beschriebenen Prozesse, wie das „Messen und Experimentieren“ (Measure and Experiment), die „Priorisierung von Arbeit“ (*Prioritize Effort*), oder Taktiken

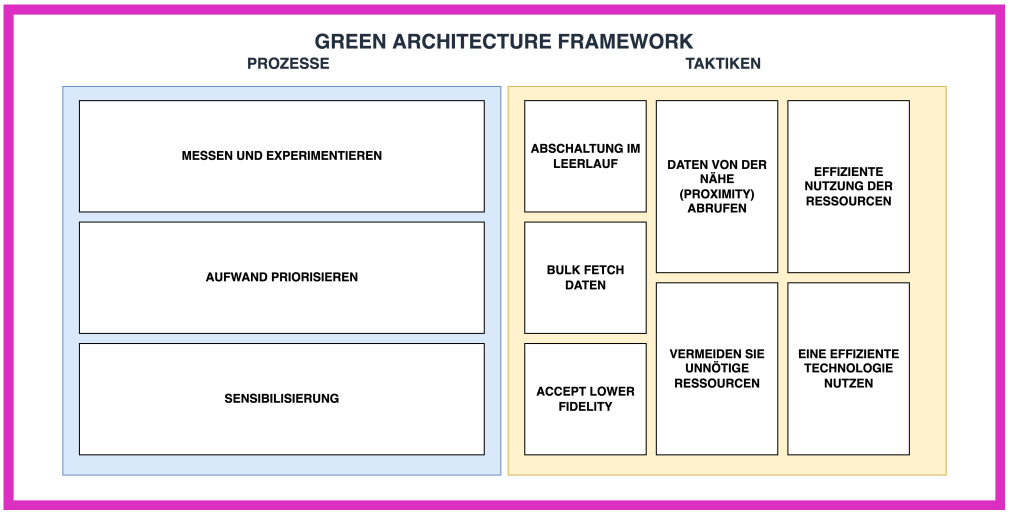

Abbildung 3.7: Green Architecture Framework[15]

wie „Herunterfahren im Leerlauf“ (*Shutdown when Idle*) oder das Abrufen von Daten aus der Nähe (*Fetch Data from the Proximity*), dienen als Bausteine, um die Energieeffizienz von Softwareprodukten systematisch zu verbessern. Durch die Anwendung dieser Prinzipien haben Entwickler die Möglichkeit, konkret ihre Systeme im Detail zu untersuchen und Lösungsstrategien zu entwickeln, die zu einer effizienteren Nutzung der Ressourcen führen.

Im Vergleich zu anderen beschriebenen Modellen, wie das GREENSOFT Modell, aus Abschnitt 3.1.3, könnte das vorgestellte „Green Architecture Framework“ als weniger komplex und ausgereift erscheinen. Ohne die Arbeit tiefer zu analysieren, zeigt ein kritischer Blick, dass Taktiken und Prozesse getrennt aufgeführt werden, jedoch oft zusammenwirken müssen. Des Weiteren fehlt eine explizite Diskussion über Metriken, die in anderen Modellen eine zentrale Rolle spielen, hier jedoch nur indirekt durch die vorgeschlagenen Taktiken adressiert werden. Trotzdem könnten die pragmatischen Empfehlungen des Frameworks Entwicklerteams konkrete Ansatzpunkte bieten, um den Energieverbrauch ihrer Software gezielter zu reduzieren.

3.1.7 Diskussion: Nachhaltigkeit in weiteren Informatik Domänen

In diesem Abschnitt werden weitere Felder der Informatik mit Nachhaltigkeit in Verbindung gebracht. Diese Abschnitte werden kurz gehalten. Nachhaltigkeit wird mit Cloud-

Computing, Virtualisierung, Netzwerke, Kommunikation und Big Data in Abschnitt 3.3 vertieft.

Grüne Software und Sicherheit in der Informationstechnik

Informationssicherheit stellt ein etabliertes Gebiet der Informatik dar, das in sämtlichen Bereichen der Informatik von zentraler Bedeutung ist. Sicherheit spielt eine Rolle bei der Entwicklung von Programmiersprachen und der Datenhaltung in Datenbanken über verteilte Systeme und Betriebssysteme bis zu Softwareentwicklungsprozessen. In diesem Kontext weisen die Konzepte der grünen Softwareentwicklung und der Informationssicherheit Parallelen auf, da beide als übergreifende Konzepte verstanden werden können. Die Gewährleistung der Systemsicherheit kann darüber hinaus als ein Aspekt der Nachhaltigkeit in der Softwareentwicklung betrachtet werden.

Informationssicherheit befindet sich oft in einem Spannungsfeld mit anderen Softwareanforderungen. Daher ist stets eine sorgfältige Abwägung erforderlich, um zu bestimmen, welches Maß an Sicherheit notwendig ist und welche Kosten für deren Implementierung in Kauf genommen werden. Sicherheitsmaßnahmen können die Benutzerfreundlichkeit, Barrierefreiheit, Leistung, Entwicklungsgeschwindigkeit und weitere Aspekte beeinträchtigen. Diese Abwägungen erfordern ein ausgewogenes Management, von dem die nachhaltige Softwareentwicklung möglicherweise Methoden adaptieren könnte. Nachhaltige Softwareentwicklung steht jedoch ebenfalls in einem Spannungsfeld zur Informationssicherheit, da Sicherheitsmaßnahmen häufig den Aufbau von Redundanzen, die Verschlüsselung von Daten und die Integration zusätzlicher Kommunikationsschichten in Protokolle und Algorithmen erfordern, was wiederum den Ressourcenverbrauch erhöht und dem Prinzip des Minimalismus entgegensteht. Auch hier ist ein sorgfältiges Management der Anforderungen und des Software-Designs erforderlich, um ein Gleichgewicht herzustellen.

Grüne KI

Künstliche Intelligenz (KI) erweitert die Grenzen der automatisierbaren Aufgaben und eröffnet damit Wege, Aufgaben zu automatisieren, die bisher mit herkömmlichen Methoden der Informatik kaum oder nicht automatisierbar waren. Dies erschließt eine neue Ebene der Digitalisierung, wobei die Komplexität der zu bewältigenden Aufgaben steigt. KI umfasst eine breite Palette an Methoden, wobei in jüngster Vergangenheit der größte

Teil der Weiterentwicklung im Bereich des maschinellen Lernens (ML) stattgefunden hat (wie Transformer, GANs und LLMs).

Um maschinelles Lernen zu betreiben, muss zunächst ein ML-Modell trainiert werden, bevor es effektiv eingesetzt werden kann. Die Genauigkeit, mit der ein ML-System Muster erkennt, hängt direkt von der Art und Weise sowie der Menge und Qualität der verarbeiteten Daten ab. Der Prozess des Datensammelns, des Trainierens von Modellen und der Nutzung trainierter Modelle erfordert erhebliche Ressourcen, einschließlich IT-Ressourcen wie CPU- und GPU-Leistung sowie Speicherkapazitäten, darüber hinaus auch natürliche Ressourcen wie Hardwarekomponenten, Energie und die Infrastruktur von Rechenzentren. Die Popularität von Large Language Models (LLMs) im Jahr 2023 hat die öffentliche Aufmerksamkeit erneut auf das Potenzial der KI, insbesondere des ML, gelenkt. Eine Vielzahl von Unternehmen sind seit dem Durchbruch von LLMs gegründet worden. Die Verwendung von KI ist somit in der Breite gestiegen, jedoch ist dies nicht an der Weiterentwicklung der Methodik im Hinblick auf Nachhaltigkeit und Ressourcen-Effizient gekoppelt.

3.2 Cloud Native Sustainability

In diesem Unterkapitel wird der Bereich Cloud Native Sustainability eingeführt. Im nächsten Abschnitt 3.3 wird dieses Feld vertieft. Dieser Abschnitt dient der Übersicht. Es wird eine Linie von Netzwerken, Rechenzentren, Cloud-Computing zu Cloud Native und schließlich Cloud Native Sustainability gezogen.

3.2.1 Netzwerke, Rechenzentren und Cloud

Die Digitalisierung hat weltweit an Bedeutung gewonnen, und der Zugang zum digitalen Netzwerk wird zunehmend als Grundrecht angesehen, das es Menschen ermöglicht, aktiv am gesellschaftlichen Leben teilzunehmen. Software stellt ein weiteres „Interface“, mit der Welt in Kontakt zu treten. Dieser Datenaustausch wird durch eine globale Vernetzung von Servern ermöglicht.

Globale Vernetzung von Servern

Ende der 1990er-Jahre wurden mit dem Aufkommen des Internets Schlüsselprotokolle wie IP (Internet Protokoll), DNS (Domain Name System) und weitere entwickelt, die das Rückgrat der heutigen digitalen Kommunikation bilden. Internet Service Provider (ISP) stellen die Verbindung für Endkunden zu diesem Netzwerk her, die dann über große Anbieter wie Google, Orange oder Cloudflare vernetzt sind und so eine Brücke zwischen verschiedenen ISPs schlagen. Diese Infrastruktur erleichtert nicht nur den Zugang zu Informationen und Diensten, sondern fördert auch nachhaltig die Vernetzung über geografische und kulturelle Grenzen hinweg, indem sie eine Plattform für globale Kommunikation und Austausch schafft.

Die globale Vernetzung ermöglicht es, Ressourcen gemeinsam zu nutzen und auf Inhalte weltweit zuzugreifen, was die Grundlage für die Digitalisierung in verschiedenen Lebensbereichen bildet. Diese Konnektivität hat die Entwicklung und Verbreitung von Cloud-Computing maßgeblich vorangetrieben. Durch Cloud-Dienste können Nutzer auf Anwendungen und Daten von überall zugreifen, was Arbeitsweisen revolutioniert und neue Möglichkeiten für die Zusammenarbeit eröffnet.

Allerdings bringt die globale Vernetzung auch Herausforderungen mit sich. Die Entwicklung komplexer, robuster, sicherer und latenzarmer verteilter Systeme ist eine Herausforderung. Die Notwendigkeit, Daten über weite Strecken mit verschiedenen Betreibern zu übertragen, erhöht die Anforderungen an die Netzwerkinfrastruktur und öffnet Sicherheitsrisiken für das Netzwerk. Zudem steigen mit der zunehmenden Vernetzung die Anforderungen an die Überwachung und das Management dieser komplexen Systeme, um Ausfälle zu vermeiden und die Systeme aufrechtzuerhalten.

Rechenzentren spielen eine Schlüsselrolle in der digitalen Wirtschaft, indem sie die notwendige Infrastruktur für die Speicherung, Verarbeitung und den Austausch von Daten bereitstellen. Sie bestehen aus physischer Infrastruktur, wie Gebäuden und Energieversorgungssystemen, und IT-Infrastruktur, einschließlich Servern und Netzwerkkomponenten. Der hohe Energieverbrauch von Rechenzentren, verbunden mit dem Einsatz natürlicher Ressourcen, einschließlich seltener Erden, steht im Mittelpunkt der Diskussion um eine nachhaltige IT-Infrastruktur. Aus Kostengründen, rechtlichen Auflagen und Nachhaltigkeitsbestrebungen werden Strategien entwickelt, um die Energieeffizienz zu verbessern und den ökologischen Fußabdruck zu verringern. Dazu gehören der Einsatz erneuerbarer Energien, verbesserte Kühltechnologien und das Design energieeffizienter Hardware.

Cloud-Computing

Cloud-Computing hat sich als ein wesentlicher Bereich innerhalb des Forschungsfeldes der verteilten Systeme entwickelt. Es bezieht sich auf die effiziente Nutzung von Servern, die in einem oder mehreren Rechenzentren zur Verfügung gestellt werden, um eine Vielzahl von Dienstleistungen und Ressourcen über das Internet bereitzustellen. Diese Technologie ermöglicht es Unternehmen und Einzelpersonen, auf eine flexible, skalierbare und kosteneffiziente Weise auf Rechenressourcen zuzugreifen, ohne die Notwendigkeit, physische Hardware vor Ort zu besitzen oder zu warten. An dieser Stelle wird vorausgesetzt, dass Begriffe wie Public Cloud, Private Cloud, Cluster und Infrastructure as a Service (IaaS) bekannt sind.

3.2.2 Cloud Native

Mit dem Aufkommen der Cloud durch Anbieter wie AWS sind neue Möglichkeiten zur Bereitstellung von Dienstleistungen entstanden. Unternehmen wie Netflix, das Video-Streaming anbietet, oder Dropbox, das Cloud-Speicherlösungen bereitstellt, sind Beispiele für diese Entwicklung. Diese Weiterentwicklung von Dienstleistungen und die damit verbundene Zunahme der Komplexität erfordern den Einsatz neuer technischer Mittel. Es entstehen Architekturen wie Service Oriented Architecture (SOA) und Microservices, und agile Softwareentwicklungsprozesse wie Scrum und Kanban gewinnen an Bedeutung. Cloud Native kann als eine der zahlreichen technischen Innovationen dieser Zeit betrachtet werden, die versuchen, zeitgemäße Lösungen für den Umgang mit Komplexität zu finden.

Der Begriff Cloud Native entstand mit der Einführung des Open-Source-Projekts Kubernetes, das auch als das Betriebssystem der Cloud beschrieben wird. Kubernetes verwaltet Applikationen, die als Pods auf Servern (Nodes) laufen. Es ermöglicht die Verwaltung hunderter Nodes mit entsprechend vielen Pods, unterstützt das dynamische Hoch- und Herunterskalieren von Pods, das Neustarten von Pods, die Durchsetzung von Netzwerkrichtlinien und vieles mehr. Durch diese Fähigkeiten werden sehr komplexe Systeme unterstützt. Um jedoch von den Vorteilen der Cloud zu profitieren, müssen das Team, das System und die einzelnen Anwendungen angepasst werden. Cloud Native beschreibt einen ganzheitlichen Ansatz, wie dies umgesetzt werden kann.

Cloud Native zielt darauf ab, Teams, Kultur und Technologie so zu strukturieren, dass Automatisierung und spezifische Architekturen genutzt werden können, um mit der Komplexität umzugehen und die Entwicklungsgeschwindigkeit zu steigern. Es geht darum, sowohl die technologische als auch die personelle Seite von Unternehmen zu skalieren. Cloud-Native-Praktiken können auch außerhalb von Cloud-Umgebungen angewendet werden[7]. Technologien wie Microservices, Containerisierung, dynamische Orchestrierung, DevOps, Skalierbarkeit, Unveränderlichkeit, Resilienz, Sicherheit, Beobachtbarkeit und Monitoring spielen zusammen, um ein Cloud-Native-System zu entwickeln.

Die Cloud Native Computing Foundation, ein Teil der Linux Foundation und eine Open-Source-Organisation, verwaltet Cloud-Native-Technologien. Mit Kubernetes ist ein Ökosystem von Projekten entstanden, die Kubernetes als Inspiration nutzen oder es ergänzen. Die CNCF veröffentlicht eine visuelle Übersicht, die die Landschaft dieser Projekte in verschiedenen Kategorien wie Continuous Integration & Delivery, Datenbanken, Streaming & Messaging, Planung & Orchestrierung, API-Gateways, Service-Proxies, Service-Meshes, Container-Runtime und mehr darstellt[26].

3.3 Cloud Native Sustainability – Diskussion der Abstraktionsschichten

In diesem Abschnitt werden die Abstraktionsschichten in einem Cloudsystem besprochen, die alle relevant sind, um ein umfassendes Verständnis von Cloud Native Sustainability zu bekommen. Wie im vorherigen Abschnitt beschrieben, umfasst Cloud Native mehr als nur die Sammlung von Open-Source-Projekten. Es steht für eine neue Generation Software für die Cloud zu entwickeln. Die Aufteilung der Verantwortung, auch in Bezug auf den Ressourcenverbrauch und die Matrizen für die Nachhaltigkeit eines Systems, wird deutlich, wenn man die Abstraktionsebenen untersucht. In der folgenden Abbildung 3.8 sind die zu diskutierenden Abstraktionsebenen dargestellt. Die Abstraktionsebenen werden nacheinander besprochen.

3.3.1 Hardware-Ebene

Auf der Hardwareebene finden sich Hardware, integrierte Software sowie APIs zur Steuerung der Hardwarekomponenten. In einem Rechenzentrum kommen zudem Industriestan-

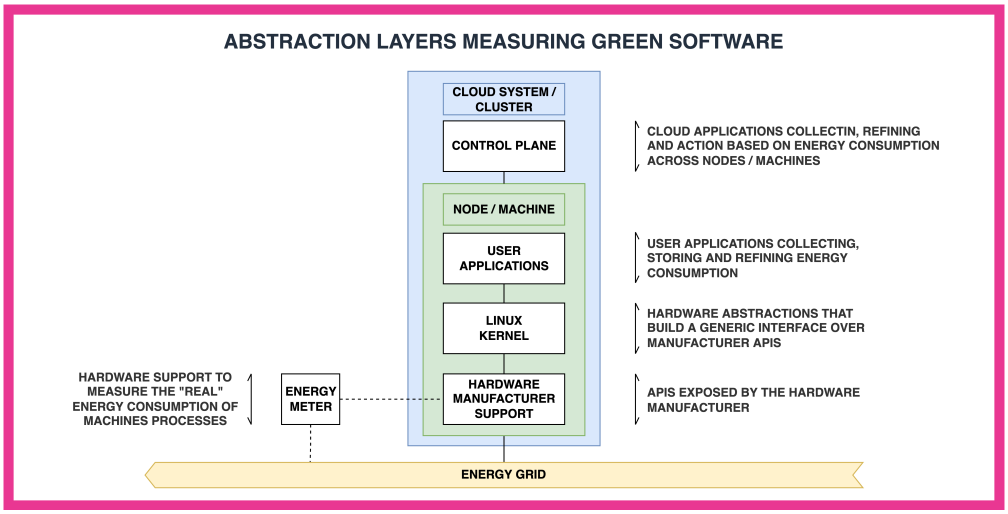

Abbildung 3.8: Abstraktionsschichten beim Messen von Grünen Software Metriken[50]

dards wie IPMI[1] (*Intelligent Platform Management Interface*) zum Einsatz, mit denen der Energieverbrauch eines gesamten Rechenzentrums überwacht und gedeckelt werden kann. Der Fokus in der Hardware-Ebene liegt auf den CPU Herstellern, die häufig auch das Überwachen anderer Komponente wie DRAM, Speicher etc. ermöglicht.

- **Intel**: Intels Chips enthalten seit der Sandy Bridge-Architektur im Jahr 2011 die Funktion „Running Average Power Limit“ (RAPL). Ursprünglich wurde RAPL entwickelt, um BIOS und Betriebssysteme in die Lage zu versetzen, virtuelle Leistungsgrenzen festzulegen, und erleichtert auch die Messung des Energieverbrauchs. Im Laufe der folgenden Iterationen wurde RAPL erweitert, um ein breiteres Spektrum an energieverbrauchenden Komponenten in verschiedenen Paketen zu erfassen. Die von RAPL abgedeckten Komponenten sind in der Abbildung 3.9 dargestellt. RAPL bietet präzise Energieverbrauchsmessungen mit einer minimalen Fehlermarge [37].
- **ARM**: ARM ist insofern einzigartig, als es die Architektur für „System-on-a-Chip-Konfigurationen“ entwirft, die Komponenten wie CPU, DRAM und Speicher integrieren. ARM baut diese Chips nicht selbst, sondern vergibt Lizenzen für die Entwürfe an Hersteller wie Qualcomm. Infolgedessen gibt es Unterschiede zwischen den Chips, unter anderem bei den Leistungsmessungsfunktionen. ARM hat das System Control and Management Interface (SCMI) eingeführt, das es den Herstel-

[1] IPMI: https://en.wikipedia.org/wiki/Intelligent_Platform_Management_Interface

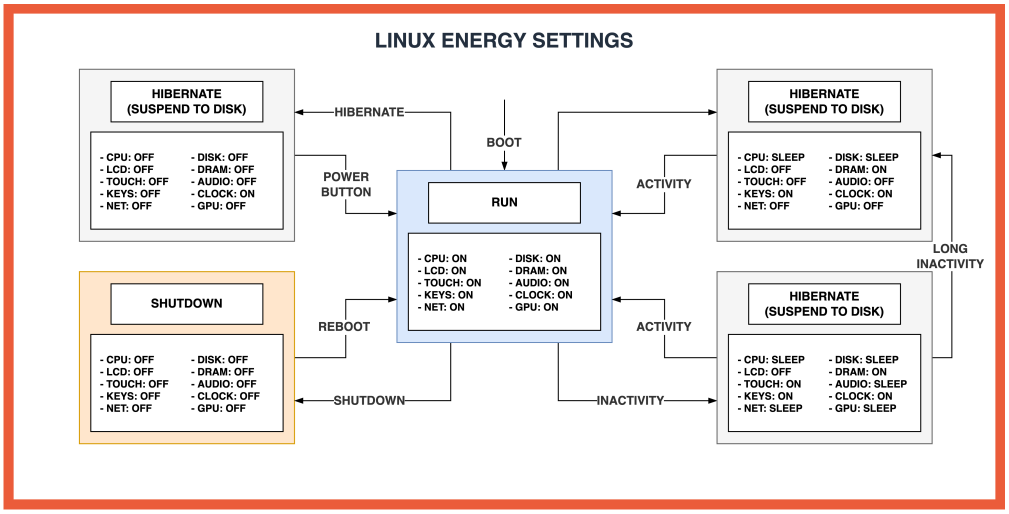

Abbildung 3.9: Von RAPL unterstützte Energiebereiche

lern von ARM-Chips ermöglicht, neben anderen Metriken auch Energiekennzahlen zu melden [2].

- **AMD**: AMD verfolgt den gleichen Ansatz wie Intel und integriert die Stromverbrauchsüberwachung in seine Chipsätze über eine RAPL-ähnliche API namens APML (Advanced Platform Management Link), wodurch die Fähigkeit zur Überwachung des Stromverbrauchs, insbesondere für energieempfindliche mobile und IoT-Geräte, verbessert wird [3]. AMD entwickelt auch Kernel-Treiber für die Berichterstattung in RAPL, siehe `amd_energy` [4]).

- **NVIDIA**: NVIDIA bietet mit dem System Management Interface (SMI) ein kommandozeilenbasiertes Tool, das die Verfolgung und Anpassung verschiedener GPU-Parameter, einschließlich des Energieverbrauchs, ermöglicht [5]. NVIDIA stellt auch eine API namens NVML zur Verfügung, die auf `smi` aufbaut [6].

Diese Industriestandards basieren auf softwarebasierten Methoden, die den Energieverbrauch annähernd bestimmen. Software-Näherungen sind zwar nützlich und kommen der

[2] ARM SCMI: https://developer.arm.com/Architectures/System%20Control%20and%20Management%20Interface
[3] APML: https://github.com/amd/esmi_oob_library/blob/master/APML_Library_Manual.pdf
[4] amd_energy: https://github.com/amd/amd_energy
[5] Dokumentation: https://developer.download.nvidia.com/compute/DCGM/docs/nvidia-smi-367.38.pdf
[6] NVIDIA SMI: https://developer.nvidia.com/nvidia-management-library-nvml

realen Zahl nahe, aber sie sind keine direkten Messungen und erfassen daher nicht das vollständige Bild des tatsächlichen Energieverbrauchs. Dies wird deutlich, wenn man sich das SMI-Toolkit von NVIDIA ansieht, siehe Abbildung 3.10.

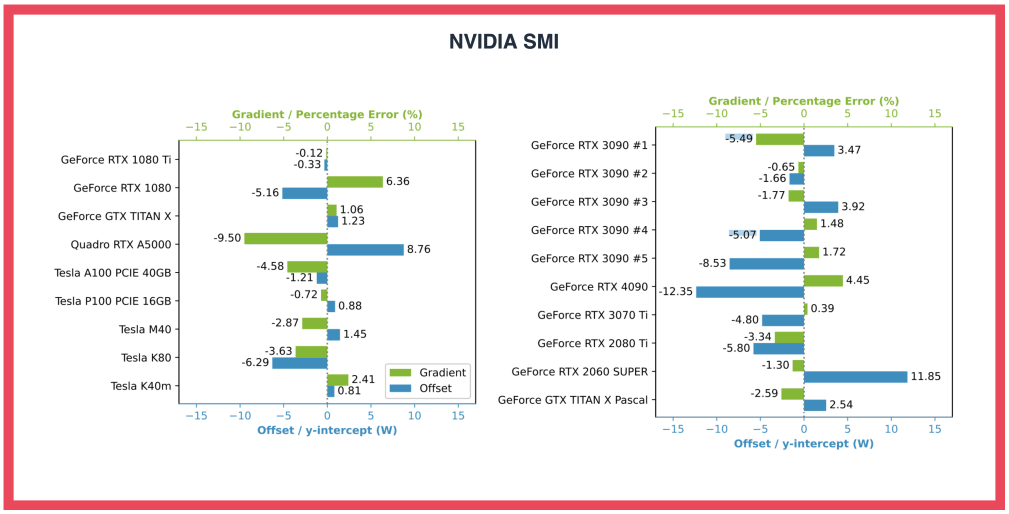

Abbildung 3.10: NVIDIA SMI Error Margin

In einer Studie vom März 2024 wurde die Genauigkeit von NVIDIAs `nvidia-smi`-Tool bei der Messung des Energieverbrauchs untersucht[72]. Die Abbildung 3.10 veranschaulicht die Notwendigkeit unterschiedlicher Offset-Kalibrierungen für jedes GPU-Modell, um die Energieverbrauchsmesswerte genau anzupassen. Weiterhin wird für jedes Modell ein Gradient angezeigt, der auf einen inhärenten Messfehler hinweist, der entweder zu einer Über- oder Unterschätzung des Energieverbrauchs führt. Dies zeigt, dass es zwar möglich ist, auf der Grundlage von Software-Näherungen präzise Energiekennzahlen zu erhalten, die Erfassung solcher Kennzahlen jedoch eine Herausforderung darstellt. Überdies können Schichten, die sich auf diese Messungen stützen, mit Messfehlern arbeiten, die in vorhergehenden Schichten entstanden sind.

Neben energiebezogenen Metriken werden in Zukunft auch Ressourcenmetriken wie Lebensdauer, erwartete Lebensdauer, natürliche Ressourcen und effektiver Nutzungsgrad (unter anderem) eine Rolle spielen. Hinzu kommt ein Fokus auf die Nutzbarkeit der Hardware für zukünftige Software-Iterationen, damit alte Hardware mit neuer Software nutzbar bleibt. Diese Metriken werden jedoch an dieser Stelle nicht weiter beleuchtet.

3.3.2 Betriebssystem-Ebene

Linux und andere Betriebssysteme (OS) sind für die Vermittlung zwischen den Anforderungen der Anwendung und den gegebenen Möglichkeiten der Hardware zuständig. Um die verfügbaren Hardwareressourcen optimal zu nutzen, muss das Betriebssystem die richtige CPU-Frequenz einstellen, CPU-Kerne aktivieren und deaktivieren und Hardwarekomponenten in den Ein-, Aus-, Ruhe- oder Leerlaufzustand versetzen, dafür sorgen, dass die Komponenten nicht überhitzen, und dem Benutzer Metriken mitteilen. Es gibt verschiedene Betriebssysteme, die jeweils andere Herangehensweisen haben, mit dieser Herausforderung umzugehen, jedoch findet Linux im Cloud-Bereich am breitesten Verwendung und inspiriert andere Betriebssysteme.

Bei der Linux-Energieverwaltung kommen zwei Hauptstrategien zum Einsatz – eine statische und eine dynamische -, die jeweils darauf abzielen, die Energieeffizienz zu optimieren und gleichzeitig unterschiedliche Betriebsanforderungen zu erfüllen:

- **Statische Energieverwaltung**: Legt von Anfang an eine feste Energiekonfiguration fest, wie die Deaktivierung ungenutzter Geräte, die nicht an die Arbeitslast des Systems angepasst wird. Dieser Ansatz gewährleistet zwar ein konsistentes Stromverbrauchsprofil, ist aber bei wechselnden Betriebsanforderungen möglicherweise nicht so effizient, da er nicht auf Änderungen der Systemaktivität reagiert.
- **Dynamische Energieverwaltung**: Ein ausgefeilterer Ansatz, der es dem Betriebssystem ermöglicht, dynamisch auf die Anforderungen des Systems zu reagieren. Dazu gehört die Anpassung der Stromversorgungszustände von Hardware-Geräten (wie *ON*, *OFF*, *STANDBY*, *SLEEP*, *HIBERNATE*, vgl. Abbildung 3.11) und die Änderung der CPU-Leistungsmerkmale durch Techniken wie Dynamic Voltage and Frequency Scaling (DVFS). Dazu gehört auch das selektive Aktivieren oder Deaktivieren von Hardwarekomponenten, wie CPU-Kernen, auf der Grundlage einer Echtzeit-Arbeitslastanalyse. Durch die Implementierung von ACPI und anderen dynamischen Verwaltungstechniken kann Linux seine Energieverwaltungsrichtlinien an den aktuellen Zustand und die Anforderungen des Systems anpassen und so den Energieverbrauch, die Leistung und die Dienstgüte (QoS) optimieren.

Es gibt verschiedene Möglichkeiten, mit den Energieverwaltungseinstellungen unter Linux zu interagieren:

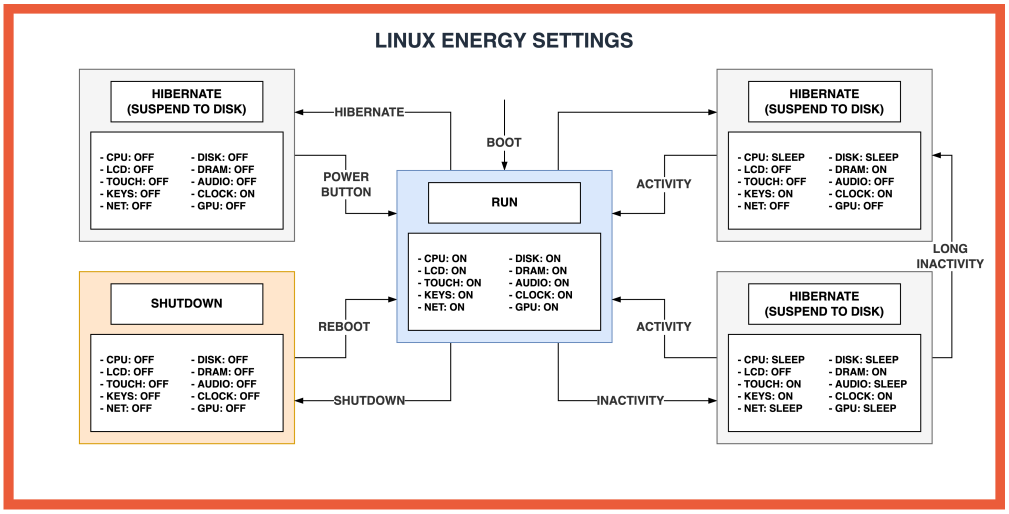

Abbildung 3.11: Energiesparende Linux-Einstellungen

- **Systemschnittstellen**: Linux stellt detaillierte System- und Hardwareinformationen sowie Steuerungsschnittstellen über spezielle Dateisysteme wie `/sys` und `/proc` zur Verfügung. Diese Verzeichnisse ermöglichen es Benutzern und Anwendungen, Systemmetriken abzufragen und Einstellungen für die Energieverwaltung vorzunehmen. So bietet beispielsweise `/sys/devices/system/cpu/cpu*/cpufreq/` Zugang zu den Einstellungen für die Skalierung der CPU-Frequenz und ermöglicht so die Kontrolle über den Stromverbrauch der CPU-Kerne.

- **Diagnosewerkzeuge**: Für die Echtzeitüberwachung und -analyse der Systemleistung und des Stromverbrauchs bietet Linux Dienstprogramme wie `top` und `htop`, die Statistiken auf Prozessebene einschließlich CPU- und Speichernutzung anzeigen. `powertop` ist speziell auf die Analyse des Stromverbrauchs zugeschnitten. Weiterhin liefert `cpufreq-info` detaillierte Informationen über die Skalierung der CPU-Frequenz, einschließlich der verfügbaren Gouverneure (verschiedene Strategien für die Skalierung der CPU-Frequenz) und der aktuellen Frequenzeinstellungen. Diese Tools eignen sich hervorragend zur Identifizierung stromhungriger Prozesse und zur Feinabstimmung der Systemleistung und Energieeffizienz.

- **Perf**: Linux unterstützt `perf`, ein leistungsfähiges Toolset zur Leistungsanalyse, das energiebewusste Profiling-Funktionen enthält. Perf kann eine breite Palette von Hardware- und Software-Ereignissen überwachen und bietet Einblicke in das Systemverhalten, die zur weiteren Optimierung des Energieverbrauchs genutzt werden können. Perf-Ereignisse werden im nächsten Artikel erneut erwähnt, wenn es

darum geht, das Konzept der Verwendung von eBPF zur Verbesserung der Energieverbrauchsüberwachung zu untersuchen.

3.3.3 Benutzeranwendungsebene

Auf der nächsten Ebene gibt es Benutzeranwendungen, die Metriken aufzeichnen und sie mit anderen Systemdaten wie der CPU-Auslastung oder Prozessinformationen kombinieren, um qualitativ hochwertigere Daten zu erzeugen. Da auf der Benutzerebene Metainformationen über die Aktivitäten einzelner Anwendungen verfügbar sein können, ist es möglich, zusätzliche Entscheidungen an das Betriebssystem weiterzugeben, um Anwendungen zu unterbrechen, Ressourcen zu sparen und sie zu einem anderen ausgewählten Zeitpunkt wieder aufzunehmen.

Eine Möglichkeit, die Landschaft zu betrachten, wäre, sie in zwei Haupttypen von Anwendungen zu kategorisieren: solche, die Daten über Energie sammeln, und solche, die die gesammelten Daten nutzen. Neben diesen Anwendungen gibt es Methoden, die APIs, Standards und Patterns definieren, die uns helfen, unsere Anwendungen besser zu verbinden. Diese Landschaft ist in Abbildung 3.12 dargestellt.

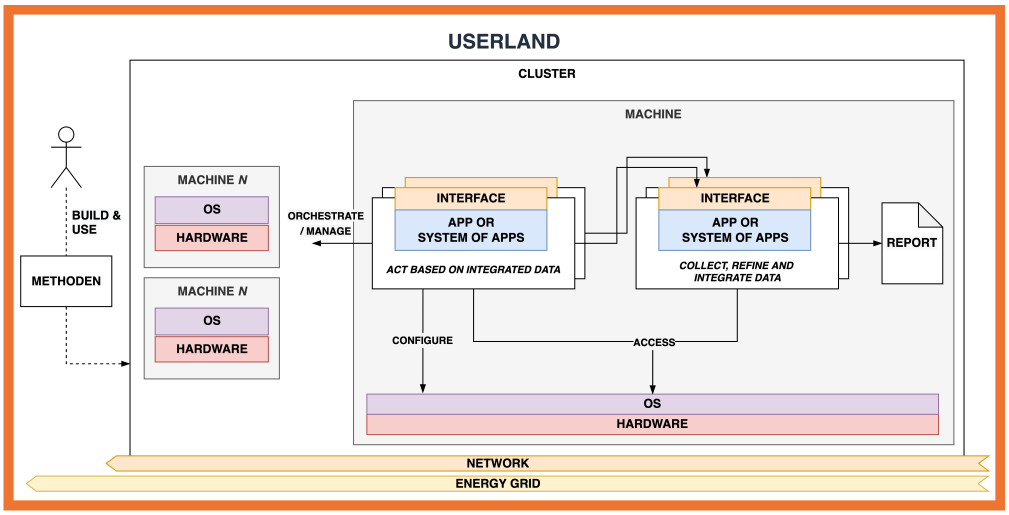

Abbildung 3.12: Linux Userland Struktur

- **Methodologien**: Dies sind die Rahmenwerke und Ansätze, die festlegen, wie Anwendungen entwickelt werden, einschließlich der APIs, die die Kommunikation zwischen verschiedenen Softwarekomponenten erleichtern.

- **Anwendungen, die Daten sammeln**: Diese Anwendungen oder Systeme von Anwendungen konzentrieren sich auf das Sammeln von Informationen von verschiedenen Endpunkten. Ein gängiger Anwendungsfall ist die Erfassung des Energieverbrauchs des Systems und die Schätzung des CO_2-Fußabdrucks auf der Grundlage des Standorts und des Energiemixes. Das Sammeln von Daten, die Verfeinerung der Daten und die Erstellung von Daten mit höherem Wert ist das Ziel von Anwendungen dieser Kategorie. Das Projekt Scaphandre beispielsweise integriert verschiedene APIs wie RAPL, um einen umfassenden Überblick über Energiemetriken zu erhalten. Diese Metriken werden verfeinert und in verschiedenen Formaten über einen verdaulichen Endpunkt zur Verfügung gestellt, was die Integration mit anderen Projekten wie Prometheus und Grafana erleichtert.
- **Anwendungen, die auf Basis von Daten agieren**: Diese Kategorie umfasst Anwendungen, die die gesammelten Daten verarbeiten und nutzen, um automatisierte Entscheidungen zu treffen, den Betrieb zu optimieren und die Energieeffizienz zu verbessern. Da das Betriebssystem die Hauptlast der Vermittlung des Energieverbrauchs von Anwendungen übernimmt, ist diese Aufgabe bereits erledigt und muss nicht von Anwendungen erfüllt werden. Es ist zwar möglich, das Betriebssystem über die Anwendungen zu konfigurieren, aber ich habe bisher nicht viele Projekte gesehen, die das getan haben. Daher sind die Anwendungen, die Entscheidungen treffen, in der Regel für eine andere Abstraktionsschicht konzipiert, die für das Betriebssystem unerreichbar ist – die Cloud.

In der Benutzeranwendungsebene kommen Virtualisierungstechniken zum Einsatz, um Hardware-Ressourcen effektiv zwischen Applikationen zu teilen. Virtualisierungsebenen können entweder durch Hypervisoren und virtuelle Maschinen oder durch Containerisierung implementiert werden. Wird die Maschine von einem Anbieter verwaltet und es wird lediglich ermöglicht, Applikationen auf einer Virtualisierungsebene einzusetzen, kann dies dazu führen, dass nicht alle Metriken und Daten, die eigentlich einsehbar wären, zugänglich sind. Diese Einstellungen können auf jeder Maschine vorgenommen werden.

Eine neuere Technologie, die sich neben virtuellen Maschinen und Containern einreiht, ist WebAssembly (WASM). WASM ist eine ursprünglich für den Browser entwickelte Technologie, die auch im Cloud-Bereich von großem Interesse ist. Sie ermöglicht eine effiziente und sichere Ausführung von Anwendungen in einer plattformunabhängigen, kompakten Form, die von einer Laufzeitumgebung ausgeführt wird. WASM bietet eine Alternative zu virtuellen Maschinen (VMs) und Containern, da es kleinere Binärdateien

und plattformübergreifende Kompilierung ermöglicht, was die Verteilung und Ausführung erleichtert. Zudem implementiert WASM Sandboxing für zusätzliche Sicherheit. WASM ist damit eine vielversprechende Ergänzung zu bestehenden Technologien und erweitert das Ökosystem der Anwendungsentwicklung und -bereitstellung.

3.3.4 Cloud-Ebene

In der Cloud legt der Fokus auf die Orchestrierung von Anwendungen auf zahlreichen Rechnern. Auf dieser Ebene geht es unter anderem um Skalierbarkeit, Ausfallsicherheit und Latenz. Um Anwendungen effizient auf verschiedene Hardware-Ressourcen aufzuteilen, werden auf Virtualisierungstechniken zurückgegriffen. Dieser Ansatz ermöglicht die Virtualisierung von Komponenten wie CPUs, Speicher und Netzwerkressourcen (et al.). Die zuvor besprochenen Technologien wie virtuelle Maschinen, Container und WASM haben hier besondere Bedeutung, da sie die Cloud erst ermöglichen. Diese Virtualisierungstechniken vereinfacht das Teilen von Ressourcen zwischen Nutzern, was die Auslastung verbessert und den Energieverbrauch senkt.

Kubernetes vereinfacht zusammen mit anderen Orchestertoren diese Komplexität innerhalb der Cloud-Umgebung. Vielleicht haben Sie schon einmal gehört, dass Kubernetes als das Betriebssystem der Cloud betrachtet werden kann. Kubernetes erleichtert die Interaktion zwischen Infrastrukturfunktionen und Plattformanforderungen. Wie Linux hat Kubernetes Praktiken und Schnittstellen entwickelt, die sich zu Standards entwickelt haben, und ein Ökosystem von Tools aufgebaut, die seine Funktionalitäten erweitern. Diese Cloud-Betriebssysteme verteilen Anwendungen effizient auf die verfügbaren Ressourcen, skalieren Anwendungen als Reaktion auf schwankende Anforderungen, verwalten die Konnektivität von Diensten und konfigurieren Ressourcen so, dass sie einem erwarteten Zustand entsprechen, um nur einige Aufgaben zu nennen. Was hat das mit dem Energieverbrauch zu tun?

- **Skalierung**: Bei der Skalierung geht es im Wesentlichen um die Anpassung der Anwendungskapazität an den aktuellen Bedarf, was für den Energieverbrauch eine wichtige Rolle spielt. Eine effiziente Skalierung minimiert die unnötige Ressourcennutzung und spart dadurch Energie. KEDA[7] ermöglicht etwa die Skalierung von Bereitstellungen auf 0, wodurch der Ressourcenverbrauch minimiert wird. Oder

[7] KEDA: https://keda.sh/

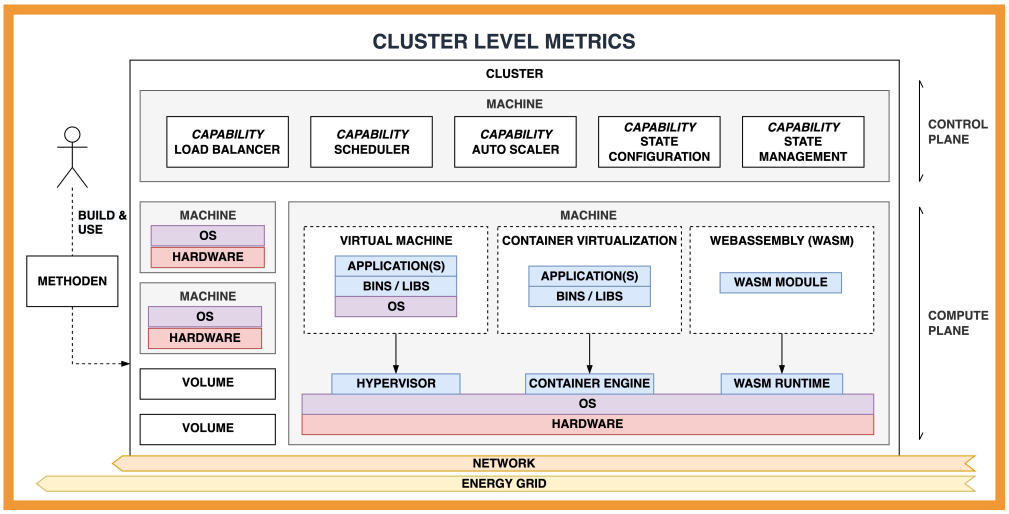

Abbildung 3.13: Cluster Level Structure

Karpenter[8], mit dem Kubernetes nicht nur Pods, sondern auch Nodes skalieren kann.

- **Scheduling**: Beim Scheduling geht es darum, Aufgaben oder Anwendungen den am besten geeigneten verfügbaren Ressourcen zuzuweisen. Sie wirkt sich auf den Energieverbrauch aus, indem sie sicherstellt, dass die Arbeitslasten so auf die Knoten verteilt werden, dass die Ressourcenauslastung maximiert und die Verschwendung minimiert wird. Anwendungen, die viel miteinander kommunizieren, können nebeneinander platziert werden, oder ressourcenintensive Aufgaben können ausgeführt werden, wenn ein Überschuss an Energie verfügbar ist (was bei erneuerbaren Energien der Fall ist). Bei der Zeitplanung geht es auch darum, neue Versionen von Anwendungen dann bereitzustellen, wenn es erforderlich ist, und nicht ständig. Es sollte nicht notwendig sein, jeden Tag neue Softwareversionen bereitzustellen (für die große Mehrheit der Systeme). KubeGreen[9] wäre ein Beispielprojekt, das sich auf energiebewusstes Scheduling konzentriert.

- **Konfiguration und Tuning**: Bei diesem Aspekt geht es um die Anpassung von Systemeinstellungen und Anwendungsparametern, um eine optimale Leistung und Ressourcennutzung zu erreichen. Es geht darum, die Knoten mit Anwendungen so einzurichten, dass die Steuerungsebene die wichtigsten Metriken kennt und die auf dem Rechner bereitgestellten Anwendungen einen minimalen Fußabdruck haben.

[8]Karpenter: https://karpenter.sh/
[9]KubeGreen: https://kube-green.dev/

Verringerung der Größe des Container-Images, Verwendung von ARM anstelle von x86, intelligente Aktualisierung von Abhängigkeiten.

4 Carbonaut Architekturdokumentation

Carbonaut ist eine Anwendung, die Daten sammelt, um einen ganzheitlichen Überblick über die Nachhaltigkeit der betriebenen Cloud Infrastruktur zu geben. Dieser Architekturüberblick dient dem Verständnis wichtiger Designentscheidungen. Die Architekturdokumentation basiert auf der Struktur der arc42[1].

4.1 Einleitung und Zielsetzung

Dieser Abschnitt führt in das Projekt ein und skizziert die Ziele, die Carbonaut verfolgt.

4.1.1 Projektüberblick

Carbonaut ist eine Applikation, die in einer Cloudumgebung eingesetzt wird, um Informationen über den Nachhaltigkeitsfußabdruck der eingesetzten IT-Infrastruktur zusammenzutragen. Carbonaut greift dabei auf existierende spezifizierte Schnittstellen in der verwalteten Infrastruktur zurück und setzt die Informationen miteinander in Zusammenhang, um damit ein holistischen Überblick zu präsentieren. Die gesammelten Informationen werden in verschiedenen Formaten zur Verfügung gestellt und können außerhalb Carbonaut's manuell oder automatisch weiterverarbeitet werden.

Carbonaut löst damit die folgenden Probleme (Features):

- Integrieren heterogener Datenschemata.
- Aufbereiten und in Zusammenhang bringen aggregierter Daten.
- Zurverfügungstellung einer zentralen Schnittstelle, um einen Überblick über den Nachhaltigkeitsfußabdruck des gesamten Systems zu erhalten.

[1] ARC42-Softwarearchitektur: https://arc42.org/

Qualitätsziel	Motivation und Erläuterung
Einheitliches aussagekräftiges Datenschema (Interoperabilität)	Das System verarbeitet heterogene Datenschemata, die in ein einheitliches Schema übertragen werden, müssen, um effektiv weiterverarbeitet werden zu können.
Plattformunabhängigkeit (Kompatibilität)	Das System ist plattformunabhängig aufgebaut, um die Funktionalität für Anbindung heterogener Infrastrukturen zu bieten.
Benutzerfreundlichkeit (Benutzbarkeit)	Die Bedienung des Systems ist an allgemeine Best Practices und Richtlinien angepasst, was die Hürde für den Einsatz von Carbonaut in komplexen Systemen verringert.
Ressourcenverbrauch (Effizienz)	Das System hat einen minimalen Ressourcenverbrauch.
Datenaktualität (Funktionale Eignung)	Das System stellt sicher, dass die Daten in Echtzeit oder in regelmäßigen Intervallen aktualisiert werden, um aussagekräftig zu sein.
Modularität (Erweiterbarkeit)	Das System ist modular aufgebaut und kann von den Entwicklern individuell angepasst werden.

Tabelle 4.1: Carbonaut übergreifende Qualitätsziele

- Dynamisch erweiterbare, das zur Laufzeit auf Topologie Veränderungen der IT-Infrastruktur anspringt und die Datenakquise dementsprechend anpasst.
- Anbindung benutzerdefinierter APIs, um Informationen zusammenzutragen.

4.1.2 Qualitätsziele

Die folgende Tabelle 4.1 beschreibt die zentralen Qualitätsziele von Carbonaut, wobei die Reihenfolge eine grobe Orientierung bezüglich der Wichtigkeit vorgibt.

Die Qualitätsszenarien in Abschnitt 4.10 konkretisieren diese Qualitätsziele.

4.1.3 Stakeholder

Die folgende Tabelle 4.2 stellt die Stakeholder von Carbonaut und ihre Interessen dar.

Stakeholder	**Interesse / Bezug**
Cloud-Anbieter und IT-Infrastruktur -Dienstleister	Integration von Nachhaltigkeitsdaten in ihre Dienste, um ihren ökologischen Fußabdruck zu verringern und nachhaltigere Lösungen anzubieten.
Unternehmen und Organisationen (*Adopter*)	Nutzung der konsolidierten Daten zur Verbesserung ihrer eigenen Nachhaltigkeitsstrategien und zur Einhaltung von Umweltvorschriften und -standards.
Technologieanbieter und Softwarehersteller	Integration der Carbonaut-Daten in ihre Produkte und Lösungen, um ihren Kunden nachhaltigere IT-Optionen anbieten zu können.
Open-Source-Community und Entwickler	Beitrag zur Weiterentwicklung des Projekts, um nachhaltigere Softwarelösungen zu fördern und eigene Kenntnisse und Fähigkeiten zu erweitern.
Regulierungsbehörden und staatliche Institutionen	Zugriff auf genaue Daten zur Überwachung und Durchsetzung von Umweltvorschriften und zur Entwicklung neuer Richtlinien für nachhaltige IT-Infrastrukturen.

Tabelle 4.2: Carbonaut Stakeholders

4.2 Randbedingungen

Beim Lösungsentwurf sind zu Beginn verschiedene Randbedingungen zu beachten, welche sich auf die Lösung fortwirken. Dieser Abschnitt stellt sie dar und erklärt auch – wo nötig – deren Motivation.

4.2.1 Technische Randbedingungen

Die folgende Tabelle 4.3 stellt die technischen Randbedingungen auf.

4.2.2 Organisatorische Randbedingungen

Die folgende Tabelle 4.4 stellt die organisatorischen Randbedingungen auf.

4.2.3 Konventionen

Die folgende Tabelle 6.2 stellt allgemeine Konventionen des Projektes auf.

Randbedingung	**Erläuterung / Hintergrund**
Betrieb in der Cloud	Einsatz fokussiert sich auf das, im Server-Bereich weitverbreitete Betriebssystem Linux. System ist agnostisch und kann auf jeglicher Cloud Infrastruktur eingesetzt werden.
Implementierung in Go	Einsatz in einem Go lastigen Cloud Native Umfeld. Entwicklung unter Version Go v1.22. Das Projekt soll auch auf neueren Go-Versionen, sobald verfügbar, laufen.
Geringe Hardwareauslastung	Betrieb der Lösung auf einem marktüblichen Standard Linux basierenden VM.
Fremdsoftware frei verfügbar	Falls zur Lösung Fremdsoftware hinzugezogen wird (z.B. API SDKs), sind diese frei verfügbar und kostenlos.

Tabelle 4.3: Carbonaut technische Randbedingungen

Randbedingung	**Erläuterung / Hintergrund**
Team	Leonard Pahlke, unterstützt durch betreuende Professoren.
Zeitplan	Beginn der Entwicklung Dezember 2023, erster lauffähiger Prototyp März 2024 (Absprache mit betreuendem Professor), vorzeigbare Version April 2024 (Absprache mit betreuendem Professor). Fertigstellung Version 1.0: 15. Juni 2024 (Abgabe der Masterarbeit).
Vorgehensmodell	Entwicklung Risikogetrieben, iterativ und inkrementell. Zur Dokumentation der Architektur kommt arc42 zum Einsatz.
Entwicklungswerkzeuge	Entwürfe mit Stift und Papier, ergänzt durch Drawio. Die Arbeitsergebnisse für die Architekturdokumentation werden auf einer eigenen Dokumentations-Website gesammelt. Die Ressourcen der Website sind auf GitHub öffentlich zugänglich[2]. Der Go-Quellcode wird in vsCode entwickelt, aber da keine entwicklungsumgebungsspezifischen Konfigurationen angegeben sind, kann die Entwicklung auch in anderen Umgebungen erfolgen.
Testwerkzeuge und -prozesse	Go Unit und Integration Tests werden für inhaltliche Richtigkeit entwickelt.
Veröffentlichung als Open Source	Die Quelltexte der Lösung wird als Open Source verfügbar gemacht. Lizenz: Apache-2.0. Gehostet bei GitHub[3].

Tabelle 4.4: Carbonaut organisatorische Randbedingungen

Konvention	Erläuterung / Hintergrund
Architektur-dokumentation	Terminologie und Gliederung nach dem deutschen arc42-Template in der Version 6.0
Kodierrichtlinien für Go	Go Coding Konventionen[4], geprüft mithilfe der Go-Toolchain und verwandten Open-Source-Projekten wie Golint
Sprache (Englisch)	Der Code wird zur vereinfachten open source Adaption in Englisch entwickelt.

Tabelle 4.5: Carbonaut Konventionen

4.3 Kontextabgrenzung

Dieser Abschnitt beschreibt das Umfeld von Carbonaut. Für welche Benutzer ist es da, und mit welchen Fremdsystemen interagiert es?

4.3.1 Fachlicher Kontext

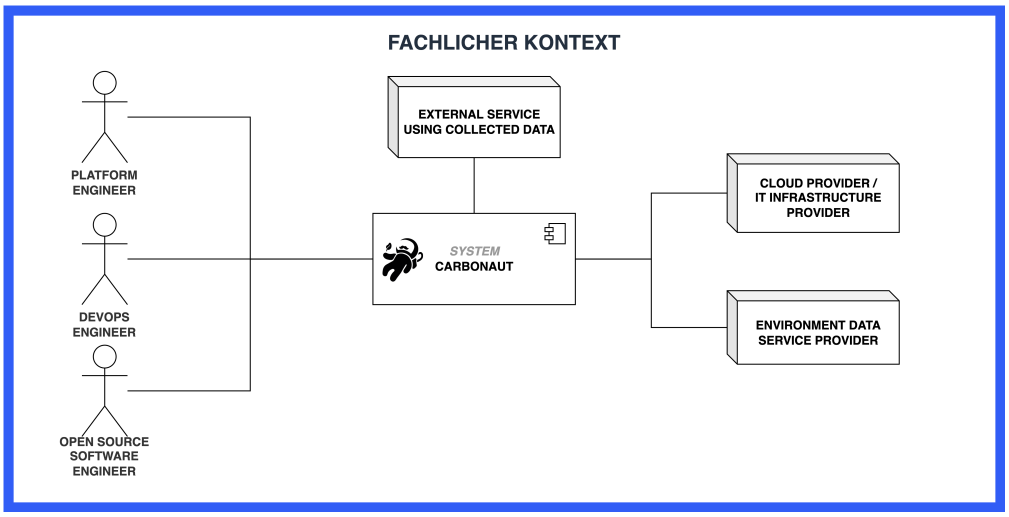

Abbildung 4.1: Carbonaut fachlicher Kontext

- **Plattform Engineer**: Verwalten des Carbonaut Deployments und der Konfiguration. Carbonaut verwaltet eine Schnittstelle, die Infrastrukturänderungen zur Laufzeit unterstützt.

- **DevOps Engineer**: Verwendung der gesammelten Carbonaut Daten zur Erstellung manueller Reports oder Einbettung in bestehende Geschäftsprozesse. Carbonaut verwaltet eine Schnittstelle zu gesammelten Daten.
- **Open-Source-Software Engineer**: Weiterentwicklung des Carbonaut Projekts.
- **External Service using the Data**: Tooling zur weiteren Veranschaulichung der gesammelten Daten, wie ein Grafana Dashboard. Carbonaut verwaltet eine Schnittstelle zu gesammelten Daten.
- **Cloud Service Provider**: Betreiber von Cloudinfrastruktur oder Bare-Metall-Rechenzentren, die APIs anbieten, um IT-Infrastruktur zu verwalten. Carbonaut hat die Funktionalität, auf diese APIs über Plugins zuzugreifen.
- **Environmental Data Service Provider**: Dienstleister, die Informationen über die Umwelt sammeln, wie über den Energiemix in einer Region oder Rohstoffaufschlüsselung / Breakdown von Hardwarekomponenten. Carbonaut hat die Funktionalität, auf diese APIs über Plugins zuzugreifen.

4.3.2 Technischer Kontext

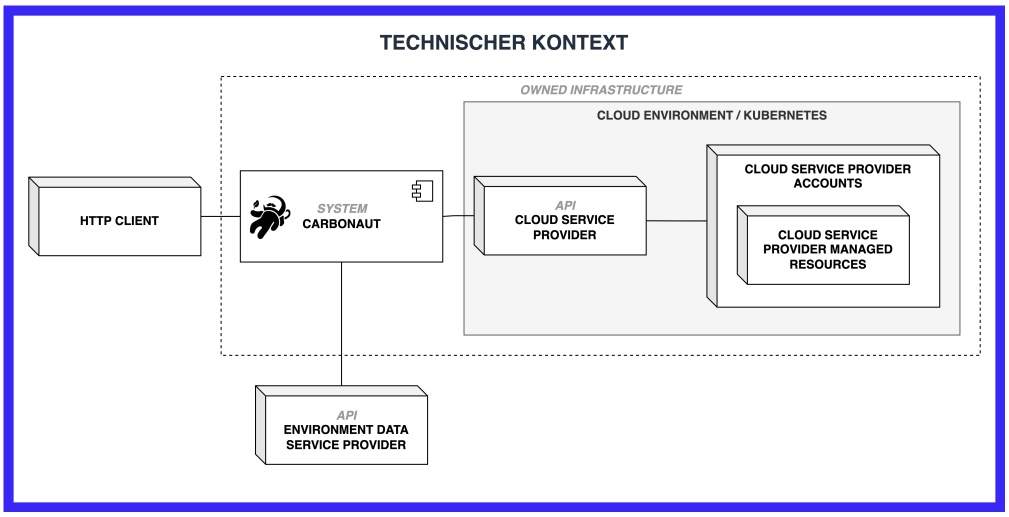

Abbildung 4.2: Carbonaut technischer Kontext

- **HTTP Endpoint**: Die Daten werden für Nutzer und externe Systeme über eine HTTP Schnittstelle bereitgestellt.

- **Environment Data Service Provider API**: Umweltdaten werden von Fremdsystemen bereitgestellt, die über Plugins in Carbonaut angebunden werden.
- **Cloud Service API**: IT-Infrastruktur-Topologie und Ressourcendaten werden über Plugins in Carbonaut angebunden.

4.4 Lösungsstrategie

Dieser Abschnitt enthält einen stark verdichteten Architekturüberblick. Eine Gegenüberstellung der wichtigsten Ziele und Lösungsansätze.

4.4.1 Einstieg

Die folgende Tabelle 4.6 stellt die Qualitätsziele von Carbonaut passenden Architekturansätzen gegenüber, und gibt damit einen Einstieg in die Lösung vor.

Die Buchstaben in Klammern → [X] verorten einzelne Ansätze aus der Tabelle 4.6 im folgenden schematischen Abbildung 4.3. Der folgende restliche Abschnitt führt in wesentliche Architekturaspekte ein und verweist auf weitere Informationen. Tiefgreifende Architekturentscheidungen werden in Abschnitt 4.9 ausgeführt.

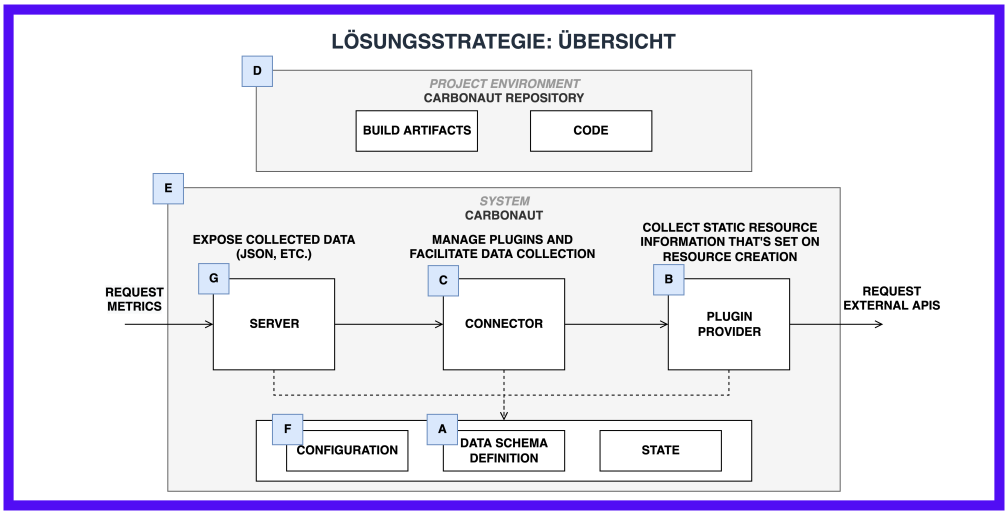

Abbildung 4.3: Carbonaut Lokalisierung der Qualitätsziele in der Architekturübersicht (vgl. Tabelle 4.6 zur Beschreibung der [X] Referenzen)

Qualitätsziel	Dem zuträgliche Ansätze in der Architektur
Einheitliches aussagekräftiges Datenschema (Interoperabilität)	• **[A]** Entwicklung eines generischen, Cloud Native Sustainability, Datenschemas. • **[B]** Heterogene Datenschemata werden über, das Adapter Design-Pattern integriert. • **[C]** Daten werden miteinander kombiniert und integriert.
Plattform-unabhängigkeit (Kompatibilität)	• **[E]**Ausführung als Go Binary oder Container. • **[D]** Öffentliche multi-arch Build Artefakte.
Ressourcen-verbrauch (Effizienz)	• **[E]** Statische, sich nicht verändernde Daten werden in einem State zwischengespeichert. • **[E]** Allokation wird intern mit der Verwendung von Pointern auf ein Minimum zu bringen. • **[E]** Das System implementiert Caching.
Modularität (Erweiterbarkeit)	• **[B]** Adapter Implementierungen können hinzugefügt werden.
Datenaktualität (Funktionale Eignung)	• **[B]** Laufzeitdaten werden pro Aufruf neu abgefragt. • **[F]** Konfigurationsparameter steuern Cache Einstellung.
Benutzer-freundlichkeit (Benutzbarkeit)	• **[F]** Das System wird über eine YAML Datei konfiguriert, die an dem Kubernetes Spec Schema angelehnt ist. • **[G]** Das System kann zur Laufzeit neu konfiguriert werden. • **[E]** Das System implementiert eine strukturierte Logging-Strategie, die von Betreibern konfiguriert werden kann. • **[D]** Das System ist öffentlich einsehbar und dokumentiert. • **[G]** Gesammelte Daten sind über eine Schnittstelle abrufbar.

Tabelle 4.6: Carbonaut Architekturansätze für Qualitätsziele (vgl. Abbildung 4.3 zur visuellen Einordnung der [X] Referenzen)

4.4.2 Aufbau

Carbonaut ist als Go-Programm realisiert. Es zerfällt grob in folgende Teile:

- die Anbindung externer Datenquellen.
- die Initialisierung der Datenakquise und Kombinierung gesammelter Daten
- das Veröffentlichen gesammelter Daten über eine Schnittstelle

Diese Dekomposition ermöglicht es, einzelne Komponenten unabhängig voneinander weiterzuentwickeln. Wird etwa ein weiteres Schema zur Darstellung der gesammelten Daten unterstützt, wird dieses von den anderen Komponenten gekapselt und beeinträchtigt die Funktionalität nicht.

Zentral für das Gelingen des Projektes ist die Anbindung externer Datenquellen und das Übertragen in ein definiertes Schema. Sowie die Effizienz des Systems, da in einem Cloudumfeld mit großen Datenmengen zu rechnen ist. Beide Konzepte werden in den beiden nachfolgenden Abschnitten vertieft. Weitere Konzepte sind in dem Abschnitt 4.8 aufgeführt.

4.4.3 Anbindung externer Datenquellen

Um gesammelte Daten zu strukturieren, wird in dem System zwischen dynamischen und statischen Daten unterschieden. Dynamische Daten verändern sich kontinuierlich zur Laufzeit, während statische Daten konsistent bleiben. Diese Struktur wird auf die benötigten Daten angewendet.

Um Aussagen über die Umweltnachhaltigkeit eines IT-Systems treffen zu können, werden sowohl Daten über die IT-Infrastruktur als auch über die Umwelt benötigt, in der die IT-Infrastruktur verwaltet wird. Die nachfolgende Abbildung 4.4 zeigt das beschriebene Datenschema des Aufbaus in Carbonaut.

In Abschnitt 4.9 wird in die detaillierte technische Abwägung eingestiegen, wie externe Datenquellen angebunden werden könnten.

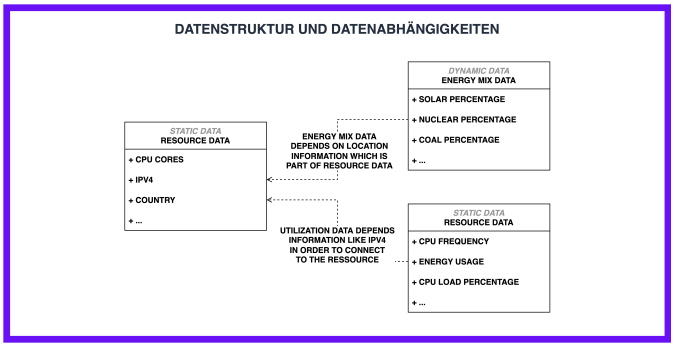

Abbildung 4.4: Carbonaut Provider Datenstruktur

4.4.4 State Management

Gesammelte Daten werden im System in ein einheitliches Schema übertragen. Die einzelnen Bausteine an Daten geben zusammen einen Überblick über das gesamte System. In der Abbildung 4.5 ist dargestellt, wie statische Daten im System strukturiert und gespeichert werden.

Der State gibt die Struktur vor, wie Daten im System verwalten werden. Dies verbessert die Effizienz, da Daten von externen Quellen nur abgefragt werden, wenn neue Informationen erwartet werden können.

Mathematische Darstellung des Datenschemas

Sei A die Menge der Accounts mit $|A| = N$,

Sei $P(a)$ die Menge der Projekte eines Accounts $a \in A$ mit $|P(a)| = N$,

Sei $R(a, p)$ die Menge der Ressourcen eines Projekts $p \in P(a)$ eines Accounts $a \in A$ mit $|R(a, p)| = N$.

Eine Ressource R_{apr} (mit $a \in A, p \in P(a), r \in R(a, p)$) hat die folgenden Daten:

Statische Ressourcen-Daten: $staticres$

CPU: CPU_{apr},

Memory: Memory_{apr},

$\vdots$

Statische Environment-Daten: $staticenv$

Region: Region_{apr},

Land: Country_{apr},

$\vdots$

Zur Laufzeit werden diese statischen Daten um dynamische Daten erweitert:

Dynamische Ressourcen-Daten: $dynres$

CPU-Frequenz: $\text{CPU\_Frequency}_{apr}(t)$,

Energieverbrauch: $\text{Energy\_Consumption}_{apr}(t)$,

$\vdots$

Dynamische Environment-Daten: $dynenv$

Solaranteil: $\text{Solar\_Percentage}_{apr}(t)$,

Kohleanteil: $\text{Coal\_Percentage}_{apr}(t)$,

$\vdots$

Hierbei bezeichnet t die Zeit, zu der die dynamischen Daten erfasst werden. Diese Darstellung fasst die Struktur der Daten und deren statische und dynamische Komponenten in einer mathematischen Formulierung zusammen. Die Struktur ist generisch formuliert und umfasst Accounts, Projekte und Ressourcen. Dadurch ist sie flexibel und eignet sich zur Abbildung verschiedener Cloud-Infrastrukturen.

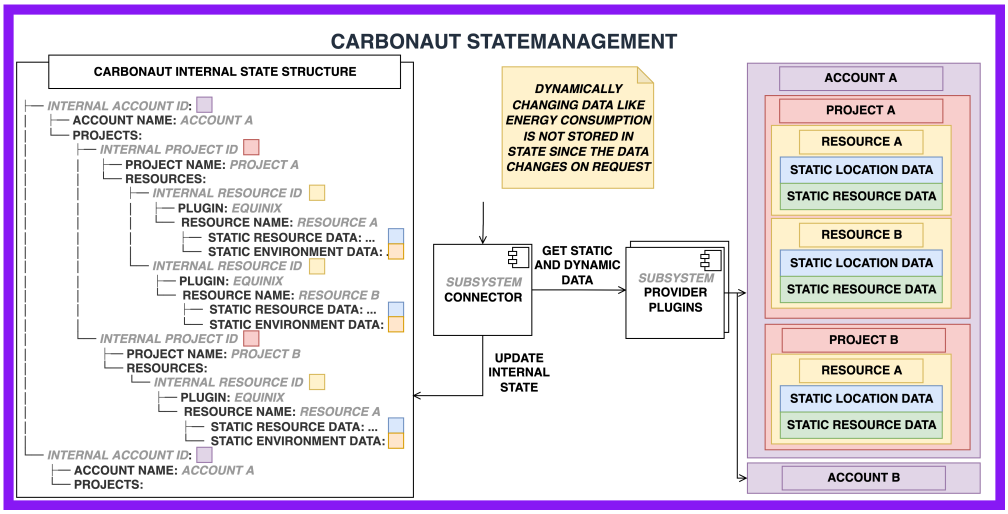

Abbildung 4.5: Carbonaut State Management

4.5 Bausteinansicht

Dieser Abschnitt beschreibt die Zerlegung von Carbonaut in Module, wie sie sich auch in der Projektstruktur des Go-Codes widerspiegelt. Module der ersten Zerlegungsebene bezeichnen wir in Carbonaut als Subsysteme. Die Bausteinsicht, in Abschnitt 4.5.1, stellt Subsysteme inklusive der Schnittstellen dar. In den darauffolgenden Abschnitten werden die beschriebenen Subsysteme ausgeführt.

4.5.1 Überblick der Subsysteme

Carbonaut zerfällt wie in Bild unten dargestellt in drei Subsysteme. Die gestrichelten Pfeile stellen fachliche Abhängigkeiten der Subsysteme untereinander dar („x → y“ für „x ist abhängig von y“). Die Kästchen auf der Membran des Systems sind Interaktionspunkte mit Außenstehenden (siehe Kontextabgrenzung in Abschnitt 4.3).

4.5.2 Server

Das Carbonaut Server Subsystem verwaltet die HTTP Schnittstelle, über die Carbonaut zur Laufzeit angesprochen werden kann. Die folgende Tabelle 4.8 zeigt die implementierten Endpunkte.

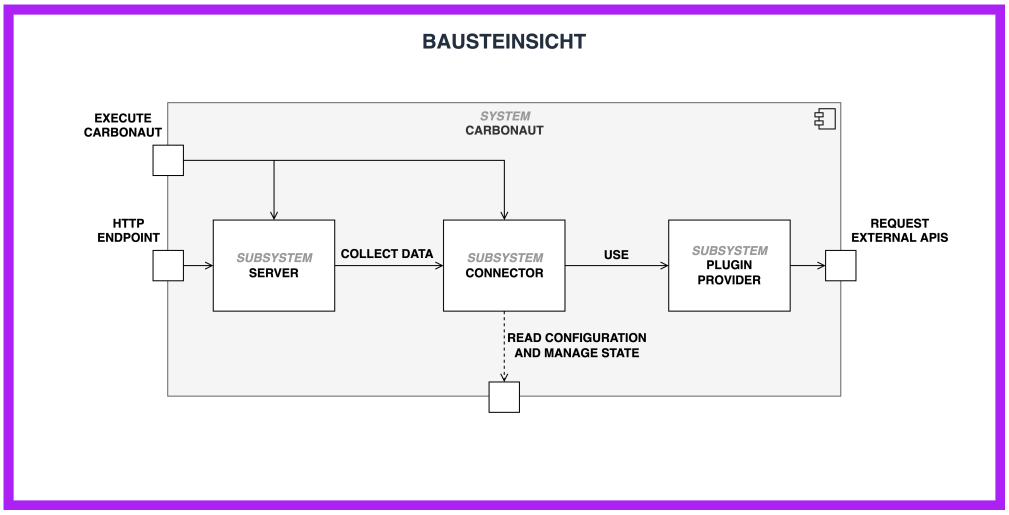

Abbildung 4.6: Carbonaut Bausteinsicht Ebene 1

Subsystem	**Kurzbeschreibung**
Server	Der Carbonaut Server HTTP Server bietet Schnittstellen, um das Deployment zur Laufzeit zu konfigurieren und gesammelte Daten abzufragen. Der Server kann gesammelte Daten in unterschiedlichen Formaten (wie JSON) exportieren. Siehe Abschnitt 4.5.2 für mehr Informationen.
Connector	Der Carbonaut Connector initialisiert Provider-Plugins und fragt Provider-Daten an. Der Connector spiegelt logisch die Infrastruktur-Topologie, initialisiert Provider-Plugins. Siehe Abschnitt 4.5.3 für mehr Informationen.
Provider-Plugins	Provider-Plugins in Carbonaut sind zur Laufzeit dynamisch eingebundene Bibliotheken, die als Adapter externe Datenquellen anbinden und die Daten in das Carbonaut-Datenschema übertragen. Siehe Abschnitt 4.5.4 für mehr Informationen.

Tabelle 4.7: Carbonaut Ebene 1 Subsysteme

Endpunkt	HTTP Type	Kurzbeschreibung
`/`	**GET**	Dieser Endpunkt gibt eine statische Nachricht zurück, welche auf API Dokumentation verweist. Dieser Endpunkt kann für HTTP „health checks“ in einer Cluster-Umgebung eingesetzt werden
`/metrics-json`	**GET**	Dieser Endpunkt fragt Daten vom Subsystem-Connector an und gibt sie im JSON-Format zurück.
`/static-data`	**GET**	Dieser Endpunkt fragt den internen State von Carbonaut an.
`/load-config`	**POST**	Dieser Endpunkt nimmt eine neue Carbonaut-Konfigurationsdatei entgegen. Die Einstellungen von Carbonaut können daher zur Laufzeit angepasst werden.

Tabelle 4.8: Carbonaut Server HTTP Endpunkte

Caching

Der Carbonaut Server speichert gesammelte Daten in einem Cache. Wie lange Daten gecacht werden, kann über die Carbonaut Konfigurationsdatei angegeben werden. Es ist anzumerken, dass in dieser Zeit trotzdem über das Subsystem *Connector* Anfragen an die IT-Infrastruktur gestellt werden können, da dieser einen parallelen Prozess beinhaltet, der neue Ressourcen in referenzierter entdeckt und logisch spiegelt (siehe folgenden Abschnitt 4.5.3.

Offene Punkte

Der Carbonaut Server veröffentlicht Daten im JSON-Format, kann jedoch erweitert werden, ohne andere Subsysteme zu beeinträchtigen. Die aktuelle Implementierung erfüllt die an Carbonaut gestellten Anforderungen. Folgende Formate und Protokolle könnten lohnende Erweiterungen sein:

- **Prometheus Exporter**: Prometheus ist ein weitverbreitetes Open-Source-Projekt in dem Cloud Native Bereich, das ähnlich wie Kubernetes ein Ökosystem an Projekten hervorgebracht hat. Die etablierten Standards (PromQL, Metriktypen etc.)

sind beispielsweise mit dem Projekt OpenTelemetry[5] standardisiert und von der Community als Referenz anerkannt.

- **gRPC[6], Thrift[7], Avro[8]**: Diese Protokolle serialisieren gesendete Daten, um eine höhere Performance zu erzielen (siehe Protobuf[9]). Die Implementierung einer oder mehrerer dieser Technologien wäre insbesondere für externe Tools, die Carbonaut als Datenquelle nutzen, von Vorteil. Diese Protokolle unterstützen zudem Data Streaming, was bei größeren Datenmengen vorteilhaft sein kann.

4.5.3 Connector

Das Carbonaut Connector Subsystem initialisiert Provider-Plugins basierend auf der angegebenen Konfiguration. Es koordiniert das strukturierte Abrufen von Provider-Daten über Plugins und integriert die gesammelten Daten in ein einheitliches Datenschema. Außerdem überwacht der Connector regelmäßig die Infrastruktur-Topologie über die Plugins, um neu erstellte Ressourcen zu erkennen und speichert dabei statische Daten in einem Zustandsspeicher ab. Die Laufzeitsicht im nachfolgenden Abschnitt 4.6 zeigt, wie dieses zentrale Subsystem im Subsystem Kontext wirkt.

4.5.4 Provider Plugins

In der Lösungsstrategie, in Abschnitt 4.4.3, wurde bereits besprochen, wie externe Datenquellen angebunden werden. Das Carbonaut Provider Plugins Subsystem beinhaltet diese beschriebenen Interfaces. Plugins implementieren wiederum Provider Interfaces und binden externe Datenquellen an. Die nachfolgende Abbildung 4.7 zeigt den Aufbau.

Im Diagramm ist zu erkennen, dass es drei verschiedene Arten von Providern gibt. Diese Provider werden in den folgenden Abschnitten 4.5.4, 4.5.4 und 4.5.4 besprochen.

[5] OpenTelemetry: https://opentelemetry.io/
[6] gRPC: https://grpc.io/
[7] Thrift: https://thrift.apache.org/
[8] Arvo: https://avro.apache.org/
[9] Protobuf: https://protobuf.dev/

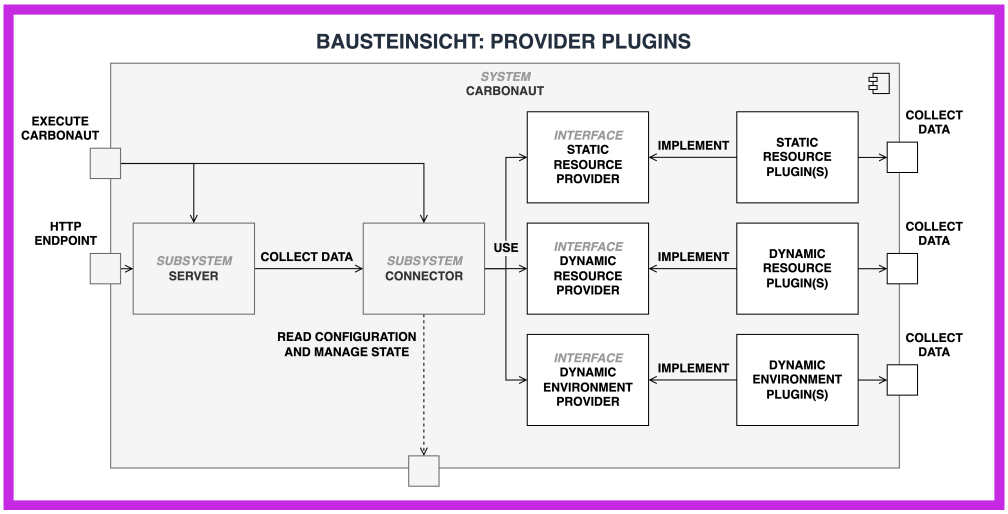

Abbildung 4.7: Carbonaut Provider Plugins Struktur

DynEnv Provider

Der *dynenv* Provider (dynamische Environment Provider) wird von Plugins implementiert, die beispielsweise Entsoe[10] oder Electricity Map[11] verwenden. Die gesammelten Daten, wie der Solaranteil in der Region Frankfurt, verändern sich von Abfrage zu Abfrage und sind daher dynamisch.

Listing 4.1: DynEnv Provider Interface (Go)

```
interface DynEnvProvider {
    GetName(): PluginKind
    GetDynamicEnvironmentData(location: Location): (DynamicEnvData, error)
}
```

DynRes Provider

Der *dynres* Provider (dynamische Ressourcen Provider) wird von Plugins implementiert, die beispielsweise Intel RAPL oder Scaphandre[12] verwenden. Die gesammelten Daten, wie CPU-Frequenz oder Energieverbrauch einzelner Prozesse, verändern sich von Abfrage zu Abfrage und sind daher dynamisch.

[10]Entsoe: https://www.entsoe.eu/
[11]Electricity Map: https://app.electricitymaps.com/
[12]Scaphandre: https://github.com/hubblo-org/scaphandre

Listing 4.2: DynRes Provider Interface (Go)

```
interface DynResProvider {
    GetName(): PluginKind
    GetDynamicResourceData(StaticResData): (DynamicResData, error)
}
```

StaticRes Provider

Der *staticres* Provider (statische Ressourcen Provider) wird von Plugins implementiert, die beispielsweise AWS[13], VMWare[14] oder Kubernetes[15] Ressourcen überwachen. Im Unterschied zu den vorherigen beiden Interfaces bietet dieses zwei weitere Methoden, um die Topologie eingesetzter Ressourcen abzufragen. Da in der Konfiguration, wie in Abschnitt 4.8.3 besprochen, nur Referenzen auf Infrastruktur-Accounts angegeben werden und keine expliziten Ressourcen, muss der *staticres* Provider Funktionalitäten implementieren, um diese Ressourcen dynamisch zur Laufzeit zu identifizieren. Die gesammelten Daten, wie Memory, Betriebssystem, CPU-Kerne und Geolocation verändern sich nicht von Abfrage zu Abfrage und werden daher statisch bei der Ressourcenerstellung abgefragt und dann zwischengespeichert (siehe State Management Abschnitt 4.4.4).

Der *staticres* Provider bildet auch statische Environment Daten ab. Dies liegt daran, dass diese Informationen mit der Erstellung von IT Ressourcen bekannt sind und daher mit der gleichen Schnittstelle angefragt werden können.

Listing 4.3: StaticRes Provider Interface (Go)

```
interface StaticResProvider {
    GetName(): PluginKind
    GetStaticResData(ProjectName, ResourceName): (StaticResData, error)
    DiscoverStaticResIds(ProjectName): ([ResourceName], error)
    DiscoverProjectIdentifiers(): ([ProjectName], error)
}
```

[13] AWS: https://aws.amazon.com/
[14] VMWare: https://www.vmware.com/
[15] Kubernetes: https://kubernetes.io/

4.6 Laufzeitsicht

Diese Sicht visualisiert im Gegensatz zur statischen Bausteinsicht dynamische Aspekte und veranschaulicht, wie die Komponenten zusammenspielen.

4.6.1 Spiegeln der IT-Infrastruktur Topologie

Der Prozess der IT-Infrastruktur Topologie Spiegelung ist in der Abbildung 4.8 gezeigt. Dieser Prozess zeigt auch die Initialisierung von Carbonaut.

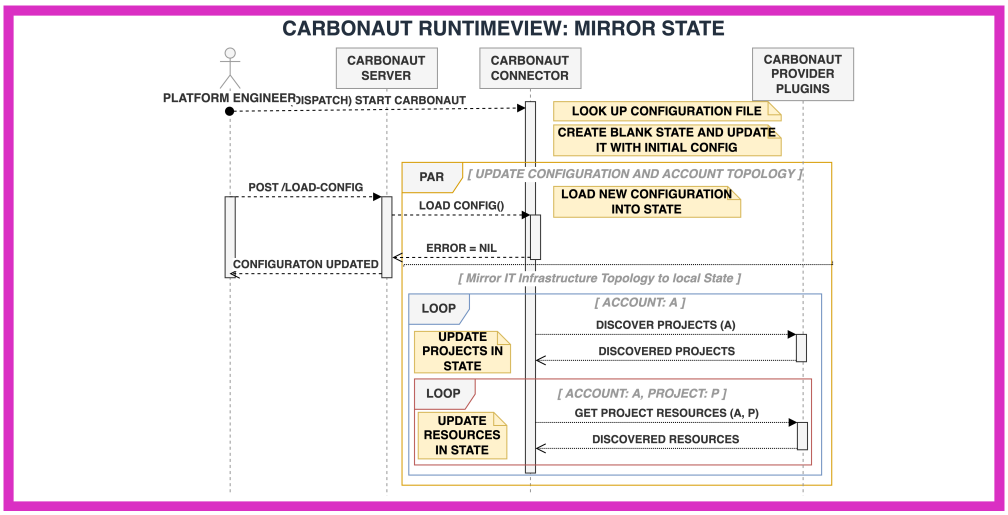

Abbildung 4.8: Carbonaut Laufzeitsicht: Mirror IT-Infrastruktur Topologie

Aus dem Diagramm geht hervor, dass zunächst nach dem Starten der Applikation die Konfigurationsdatei geladen. Dabei wird der interne State an die Konfiguration angepasst. Im nächsten Schritt wird ein weiterer Thread gestartet, um auf Konfigurationsänderungen zu reagieren. Die folgenden beiden Prozesse laufen parallel:

- **[*Parallel*] Update Configuration and Account Topologie**: Wird eine neue Konfigurationsdatei geladen, so wird der Zustand der Konfiguration angepasst. Neu erstellte Accounts werden initialisiert, gelöschte werden entfernt und die übrigen werden in den neuen Zustand übertragen.
- **[*Parallel*] Mirror IT-Infrastruktur Topology to Local State**: In regelmäßigen Intervallen werden die Projekte der Accounts abgefragt. Neue Projekte werden

angelegt, gelöschte Projekte entfernt und die übrigen Projekte werden in den neuen Zustand übertragen. In einem weiteren Loop wird über jedes Projekt in einem Account iteriert und die Ressourcen werden erfasst. Neue Ressourcen werden angelegt, gelöschte Ressourcen entfernt und die übrigen Ressourcen werden in den neuen Zustand übertragen.

4.6.2 Zusammentragen der Daten

Der Prozess der Datenakquise ist in der Abbildung 4.9 gezeigt. Dieser Prozess kann durchschritten werden, nachdem, die Applikation initialisiert ist und der Server den HTTP-Server verwaltet.

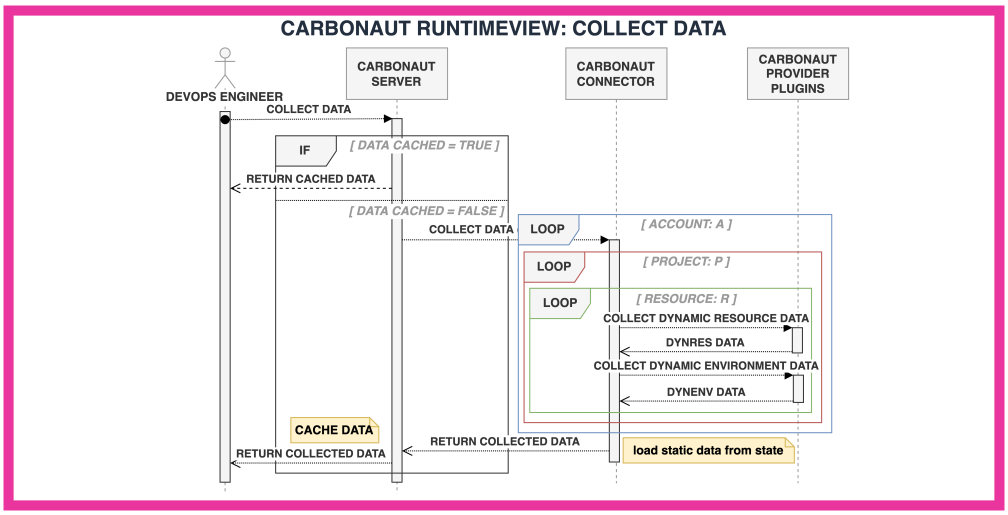

Abbildung 4.9: Carbonaut Laufzeitsicht: Collect Data

Aus dem Diagramm geht hervor, dass Anfragen über das Server-Subsystem gestellt werden können, welches entweder gecachte Daten unmittelbar zurückgibt oder diese zunächst vom Connector anfordert. Im letzteren Fall iteriert der Connector über alle Accounts, Projekte und Ressourcen und fragt die dynamischen Daten ab (siehe State Management 4.4.4).

Die gesammelten dynamischen Daten werden mit den statischen Daten kombiniert zurückgegeben. Statische Daten werden beim Spiegeln der IT-Infrastruktur-Topologie angelegt und können direkt aus dem Speicher gelesen werden (siehe vorheriges Laufzeitdiagramm in Abschnitt 4.6.1).

4.7 Verteilungssicht

Diese Sicht beschreibt den Betrieb von Carbonaut. Als Go-Programm ist es relativ anspruchslos, wenn es über das Terminal ausgeführt wird. Da Carbonaut allerdings für eine Cloud Umgebung entwickelt wurde, ist das lokale Ausführen über das Terminal eher nur in experimentellen und Entwicklungszwecken von Bedeutung. Es folgt daher eine Beschreibung, wie Carbonaut mit Containerisiert in Kubernetes eingesetzt wird.

4.7.1 Kubernetes Deployment

Die Verteilungssicht in Abbildung 4.10 zeigt den exemplarischen Einsatz von Carbonaut in einem Kubernetes-Cluster. Da Kubernetes mit Containern arbeitet, werden die im Release-Prozess erstellten Carbonaut Container-Images verwendet, die nach Docker Hub[16] gepusht werden. Über ein Deployment wird der Carbonaut-Container in einem Kubernetes Pod gestartet. Die Konfigurationsdatei wird über eine Kubernetes ConfigMap angebunden. Über einen Kubernetes Service kann auf den Carbonaut Server zugegriffen werden.

Da Carbonaut's Provider-Plugins auf API-Keys und andere Secrets in Umgebungsvariablen zurückgreifen, werden diese als Kubernetes Secrets angelegt. Überdies muss der ausgehende Datenverkehr (Egress Traffic) in den Firewall-Einstellungen des Kubernetes-Clusters freigegeben werden. Dies kann durch die Konfiguration der Network Policies in Kubernetes erfolgen oder durch Anpassungen an der Firewall des zugrunde liegenden Cloud-Anbieters. Die spezifischen Firewall-Einstellungen hängen von den verwendeten Provider-Plugins ab und müssen daher in der Carbonaut-Dokumentation nachgeschlagen werden. All diese Ressourcen werden im `carbonaut`-Namespace bereitgestellt.

Listing 4.4: Exemplarische Kubernetes Deployment YAML

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: carbonaut-deployment
  namespace: carbonaut
spec:
  replicas: 1
```

[16] Docker Hub: https://hub.docker.com/

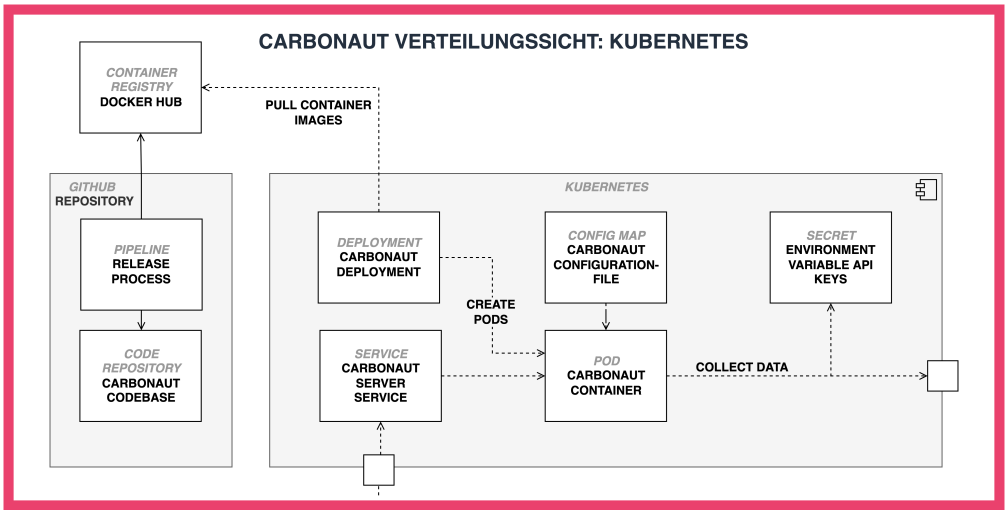

Abbildung 4.10: Carbonaut Verteilungssicht: Kubernetes

```
selector:
  matchLabels:
    app: carbonaut
template:
  metadata:
    labels:
      app: carbonaut
  spec:
    containers:
    - name: carbonaut
      image: carbonaut:latest
      ports:
      - containerPort: 8080
      envFrom:
      - secretRef:
          name: carbonaut-secrets
      volumeMounts:
      - name: config-volume
        mountPath: /etc/carbonaut
    volumes:
    - name: config-volume
      configMap:
```

```
          name: carbonaut-config
# kind: namespace ...
# kind: secret ...
# kind: config ...
# kind: service ...
```

Einsatz von Helm Charts

Helm Charts können verwendet werden, um die Bereitstellung und Verwaltung von Kubernetes-Anwendungen zu vereinfachen. Ein Helm Chart für Carbonaut kann die notwendigen Kubernetes-Ressourcen definieren. Dies ermöglicht eine konsistente und wiederholbare Bereitstellung.

Helm Charts[17] können über Konfigurationsvariablen angepasst werden. Diese Variablen können in einer `values.yaml`-Datei definiert werden, die während der Installation oder Aktualisierung des Charts übergeben wird. Image-Tags können etwa, Replica-Anzahl, Umgebungsvariablen und Ports konfiguriert werden, ohne die eigentlichen Chart-Dateien zu ändern.

4.8 Querschnittliche Konzepte

Dieser Abschnitt beschreibt allgemeine Strukturen und Aspekte, die systemweit gelten. Überdies stellt es verschiedene technische Lösungskonzepte vor. Die folgenden Abschnitte sind nicht nach Relevanz sortiert.

4.8.1 Fehlerbehandlung

Das Carbonaut-Projekt ist in der Programmiersprache Go entwickelt. Go hat eine standardisierte Vorgehensweise, um mit Fehlern umzugehen, welche auch in diesem Projekt adaptiert ist. Bei einem Funktionsaufruf wird immer ein `error` Typ zurückgegeben, wenn ein Fehler während der Ausführung auftreten kann. Beispielsweise kann bei einer Methode, die einen Integer in einen String umwandelt, kein Fehler entstehen, bei einer

[17] Helm: https://helm.sh/

Methode, die einen String in einen Integer umwandelt, jedoch schon (siehe nachfolgendes Beispiel).

Listing 4.5: Beispiel für Fehlerbehandlung in Carbonaut

```
// Kein Fehler wird propagiert, da keiner auftreten kann.
func ConvertIntToString(n int) string {
    return strconv.Itoa(n)
}

// Auftretende fehler werden propagiert.
func ConvertStringToInt(str string) (int, error) {
    result, err := strconv.Atoi(str)
    if err != nil {
        return 0, err
    }
    return result, nil
}
```

4.8.2 Logging

In Carbonaut werden strukturierte Logeinträge erstellt, die die Go-Standardbibliothek `log/slog` nutzen. Die Logeinträge werden nach Log-Ebenen sortiert (*Debug*, *Info*, *Warning*, *Error*). Die Logeinträge sind in Key-Value-Paaren organisiert, wobei der Key den Value deskriptiv beschreibt. Durch eine initiale Konfiguration wird festgelegt, wie die erstellten Logeinträge verarbeitet werden sollen. Die Logs können automatisch in eine Datei geschrieben, im Terminal angezeigt oder mit zusätzlichen Farben versehen werden.

Das nachfolgende Beispiel zeigt, wie ein Log-Eintrag generiert wird.

Listing 4.6: Generierung eines Log-Eintrags

```
slog.Debug("New Carbonaut configuration parsed",
    "unaltered_accounts", remainingAccounts,
    "deleted_accounts", toBeDeletedAccounts,
    "new_accounts", toBeCreatedAccounts,
)
```

Das nachfolgende Beispiel zeigt, wie die Konfiguration einmalig beim Start von Carbonaut gesetzt wird.

Listing 4.7: Setzen der Logging-Konfiguration beim Start

```
slog.SetDefault(slog.New(logger.NewHandler(os.Stderr, &logger.Options{
    Level:       logger.GetLogLevel(cfg.Meta.LogLevel),
    TimeFormat:  time.DateTime,
    SrcFileMode: logger.ShortFile,
})))
```

4.8.3 Konfiguration

Die Carbonaut-Konfiguration basiert auf einem Kompositionsprinzip. Es gibt eine zentrale Konfigurationsdefinition, die Konfigurationen für Subsysteme referenziert. Wenn diese Subsysteme (Server, Connector, Provider) wiederum Unterkomponenten beinhalten, werden diese ebenfalls referenziert. Auf diese Weise setzt sich die gesamte Konfiguration als Baum zusammen.

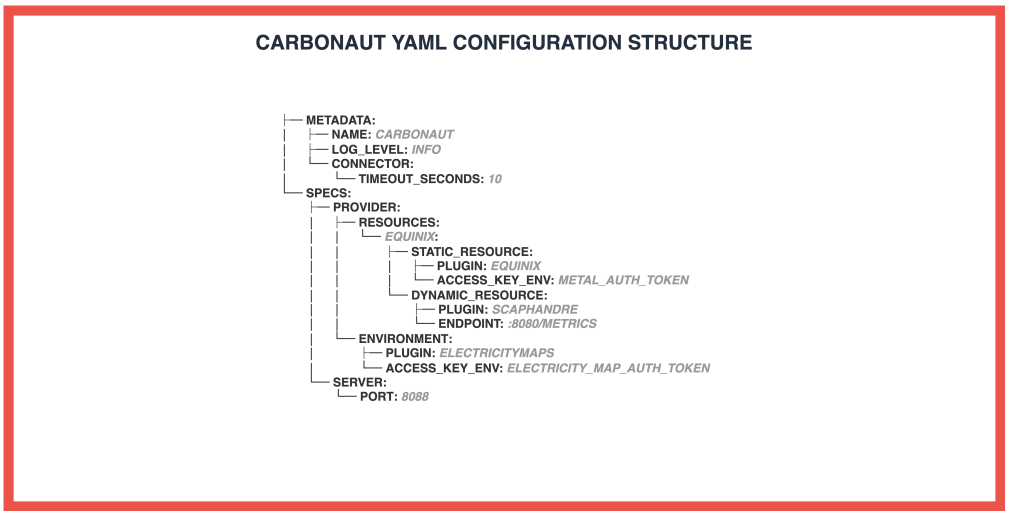

Abbildung 4.11: Carbonaut beispielhafte Konfiguration

Die zentrale Instanz, welche die Wurzel der Konfiguration hält, definiert somit nicht explizit, wie der restliche Teil der Konfiguration aufgebaut ist. Stattdessen verweist es auf die Konfigurationsdateien der Subsysteme, die ihrerseits auf die Konfigurationsdateien der Unterkomponenten verweisen.

4.8.4 Testing

Das Carbonaut-Projekt verwendet die Standard-Go-Bibliothek für das Testing. Für jede Komponente wird eine `x_test.go`-Datei angelegt. Diese Datei beinhaltet für jede öffentliche Funktion eine Testfunktion, die diese abdeckt und als Unittest prüft.

Einige Komponenten, wie die Provider Implementationen, interagieren mit externen Systemen. Um diese Funktionalität zu testen, wird auf Umgebungsvariablen zurückgegriffen, die optional gesetzt werden können, um einen End-to-End-Test (E2E-Test) der Komponente zu starten.

Einige Komponenten, wie der Connector, implementieren zusätzlich Performance-Tests. Diese Tests messen, wie viele Speicherallokationen und andere Ressourcenverwendungen die getestete Funktion durchführt.

Beim Ausführen der Tests wird automatisch eine Übersicht der Testabdeckung erzeugt, die über eine HTML-Datei visualisiert wird.

Der Anhang, siehe A.1, enthält einen Auszug, der zeigt, wie Tests, einschließlich Leistungstests, ausgeführt werden, und eine Visualisierung der Testabdeckung in einer HTML-Datei.

4.8.5 Secret Management

In Carbonaut werden API-Keys und andere sensible Informationen über Umgebungsvariablen eingelesen. Dies verhindert, dass sensible Daten explizit in der Konfiguration stehen.

4.8.6 Utility Libraries

Allgemeine Funktionalitäten, die nicht an Subsysteme gebunden sind, werden in den `util/`-Ordner verschoben. Methoden in diesem Ordner haben keine Abhängigkeiten zueinander und können daher frei verwendet werden. In Carbonaut gibt es unter anderem das `cache/`-Modul, das Caching implementiert, das `logger`-Modul, das Logging-Handler für `log/slog` bereitstellt, und das `compareutils`-Modul, das Generics verwendet, um Hilfsmethoden für Mengenoperationen anzubieten.

4.9 Architektonische Entscheidungen

Dieser Abschnitt lässt zwei zentrale Entscheidung beim Entwurf von Carbonaut im Detail nachvollziehen. Die Anbindung externer Datenquellen über Provider Plugins ist zentral für die Realisierung von Carbonaut. Ebenso wie die Entscheidung, Carbonaut als Plattform oder als einzelne Binary zu entwickeln.

4.9.1 Diskussion – Carbonaut Plattform

Zu Beginn der Architekturentwicklung stellte sich die Frage, ob Carbonaut als Plattform oder als eigenständige Komponente, die Teil eines Systems ist, entwickelt werden soll. Wie sich aus der bisherigen Architekturdokumentation erschließen lässt, wurde letzteres gewählt, jedoch bedarf dies einer Begründung, welche in diesem Abschnitt folgt. Es ist zu betonen, dass dies eine Diskussion über andere architektonische Ansätze handelt ist, welche in sich nicht im Detail ausgearbeitet sind.

Wenn man sich auf das ursprüngliche Ziel von Carbonaut besinnt, lässt sich eine serviceorientierte Architektur grob vorstellen. Diese könnte grob skizziert folgendermaßen aussehen – für jede anzubindende API wird eine Komponente gestartet, die die entsprechenden Daten sammelt. Die gesammelten Daten werden in ein Schema übertragen und in eine Message-Queue (oder einen ähnlichen Service wie Pub/Sub) gegeben. Ein weiterer Service liest nun von der Queue (oder Pub/Sub), führt die Daten zusammen und speichert sie ab. Die abgespeicherten Daten werden über eine Datenbank verwaltet und von einem API-Backend zur Verfügung gestellt. Ein Frontend visualisiert die Daten, die vom Backend bereitgestellt werden. Abbildung 4.12 zeigt den beschriebenen Aufbau.

Dieser Plattform-Designvorschlag könnte als Standard-Cloud-Vorgehensweise aufgefasst werden. Dieser Ansatz hat Vorteile und Nachteile.

Vorteile:

- Der vorgeschriebene Aufbau der Plattform erleichtert den Einsatz von Carbonaut für Anwender, da alles abgestimmt und vorgegeben ist. Frei nach dem Motto – Carbonaut ist ein ausgefeilter Service, der angeboten wird und kein Baustein, der erst im System verknüpft werden muss, um nützlich zu sein.
- Die Unterteilung in Services fördert die Skalierbarkeit.

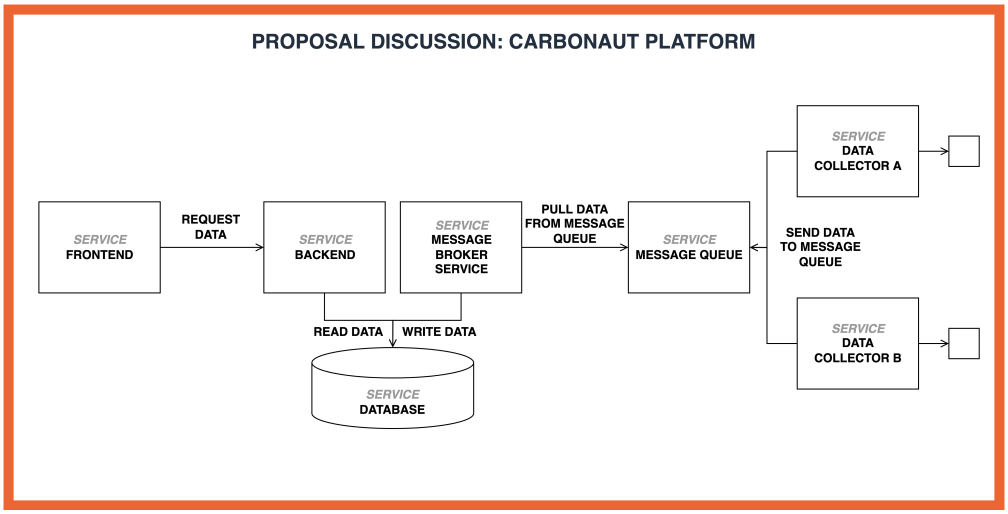

Abbildung 4.12: Carbonaut Plattform: Diskussionsgrundlage

Nachteile:

- Der Baseline-Ressourceneinsatz ist hoch, da die Unterteilung in Services ein grundlegendes Merkmal der Plattform ist.
- Eine flexible Technologiekomposition ist nicht möglich oder zumindest erschwert.
- Die Verwaltung des Systems ist selbst bei Testzwecken hochkomplex.
- Die Integration in ein bestehendes System ist erschwert, da das Projekt einen *End-to-End*-Plattform-Gedanken verfolgt.

Wie in Kapitel 3 beschrieben, verfolgt dieser Ansatz nicht den Cloud Native Gedanken. Cloud Native Systeme werden in einem Ökosystem entwickelt, das ineinander greift. Die Kernaufgabe von Carbonaut besteht nicht darin, Daten in eine Datenbank zu speichern, auch wenn das früher oder später im Prozess vermutlich geschehen wird. Die Kernaufgabe von Carbonaut ist die Integration verschiedener APIs, die Übertragung gesammelter Daten in ein einheitliches Datenschema, das Kombinieren der Daten und das Bereitstellen dieser (vgl. Abschnitt 4.1.1). Die Architektur sollte den Zielen ausgerichtet sein, was bei dem Platzformansatz nur teils zutrifft.

Ob die gesammelten Daten auf einem Grafana-Dashboard[18] oder einer Webseite visualisiert werden, ist nicht zentral für das Projekt, ebenso wenig, ob die Daten in einer

[18] Grafana: https://grafana.com/grafana/

Datenbank gespeichert oder unmittelbar verworfen werden. Diese Entscheidung wird außerhalb des entwickelten Projekts getroffen. All diese Use Cases sind denkbar, aber nicht als Ziele definiert. Carbonaut verfolgt damit einen minimalistischen, Cloud Nativen Entwicklungsansatz und wird als Baustein eines größeren Systems verstanden.

4.9.2 Diskussion – Provider Plugins

Die Anbindung externer Datenquellen an Carbonaut ist eine weitere zentrale Fähigkeit. In der Architekturdokumentation wurde beschrieben, wie Provider-Plugins als Teil von Carbonaut entwickelt werden, die als Modul / Paket importiert werden. Das Provider-Interface gibt vor, wie Plugins entwickelt werden können. Dementsprechend werden Provider-Plugins mit Carbonaut als eine Binary zu compiliert. Da die Anzahl implementierter Adapter, sowie deren Komplexität überschaubar ist, fallen keine großen Kosten an. Im Laufe der Architekturüberlegungen wurden zwei weitere Optionen diskutiert, die nun beschrieben und miteinander verglichen werden.

Architekturansatz – Anbindung über Subprozesse

Dieser Designvorschlag basiert auf der Idee, dass der Hauptprozess von Carbonaut Subprozesse starten kann, die jeweils einen Adapter darstellen. Provider-Plugins werden damit von der Carbonaut Binary getrennt. Carbonaut würde die Prozessverwaltung übernehmen, was durch die Bibliothek [19] realisiert werden kann.

Die Carbonaut Konfiguration würde auf die Provider-Plugin Binary verweisen, die entweder lokal gespeichert ist oder remote heruntergeladen wird. Carbonaut würde diese Binary als Prozess starten. Die Kommunikation zwischen Carbonaut und Provider-Plugin würde über gRPC durchgeführt werden.

Diese Lösung bietet folgende Vorteile:

- Der Adapter würde als eigenständige Binary gestartet, was bedeutet, dass er nicht zwingend in Go entwickelt sein muss.
- Ausfälle von Plugins würden den jeweiligen Subprozess abstürzen lassen, ohne den Hauptprozess von Carbonaut zu beeinträchtigen.

[19] go-plugin: https://github.com/hashicorp/go-plugin

- Plugins implementieren einen gRPC-Server, was eine generische Nutzung der Adapter auch außerhalb des Carbonaut-Kontexts ermöglicht.

Allerdings gibt es auch einige Nachteile:

- Erhöhte Komplexität durch die Verwaltung von Subprozessen und Binaries. Auch wenn die Bibliothek viel Komplexität abnimmt, ist der Aufwand höher als bei in Carbonaut kompilierten Adaptern.
- Geringere Performance im Vergleich zu Shared Libraries, da Subprozesse verwaltet werden müssen.
- Übergreifende Standards wie das Carbonaut-Datenschema müssten als separates Projekt veröffentlicht und für andere Programmiersprachen entsprechend übersetzt werden, wenn Plugins in verschiedenen Sprachen implementiert werden sollen.
- Änderungen an API und Datenschema sind verzögert zu propagieren und zu testen, da Adapter nicht direkt mit Carbonaut verbunden sind, sondern auf API-Schemas aufbauen. Carbonaut benötigt daher einen stabilen Release-Prozess und ein stabiles API-Schema.

Insgesamt ist diese Option vorteilhaft, wenn das Projekt viele Nutzer hat und dadurch die Unterteilung technische und organisatorische Vorteile ausspielen kann. Da jedoch Carbonaut nicht den dafür nötigen Reifegrad hat und auch keine ansatzweise relevanten Nutzeranzahl fallen Vorteile nicht ins Gewicht und Schwächen hindern die weitere Projektentwicklung.

Eine Anekdote: Im Open-Source-Projekt Kubernetes wurde ein ähnliches Verfahren angewandt, bei dem Komponenten mit zunehmender Reife aus dem Kernprojekt herausgelöst und über Schnittstellen extern verwaltet wurden. Ein Beispiel dafür ist die Container-Storage-Schnittstelle (CSI). Carbonaut könnte einen ähnlichen Weg einschlagen, wenn die Nutzung im Laufe der Zeit zunimmt.

Architekturansatz – Anbindung über WASM Module

Eine weitere Option für die Verwaltung von Adaptern basiert auf WebAssembly (WASM). WASM wurde in Abschnitt 3.3.3 eingeführt. In diesem Entwurfsvorschlag würde Carbonaut, WASM Module verwalten, die jeweils Provider Plugins abbilden. Dieser Ansatz ähnelt dem zuvor besprochenen Ansatz, um Prozessen.

Da der Einsatz von WASM einige andere Technologien verwendet, neben WASM im speziellen `Rust` und die Bibliothek `wasmtime`, wurde dieser Ansatz in einem experimentellen Projekt technisch ausgearbeitet. Das Projekt ist auf GitHub[20] veröffentlicht und wird nun kurz besprochen. In der Abbildung 4.13 ist der Aufbau gezeigt.

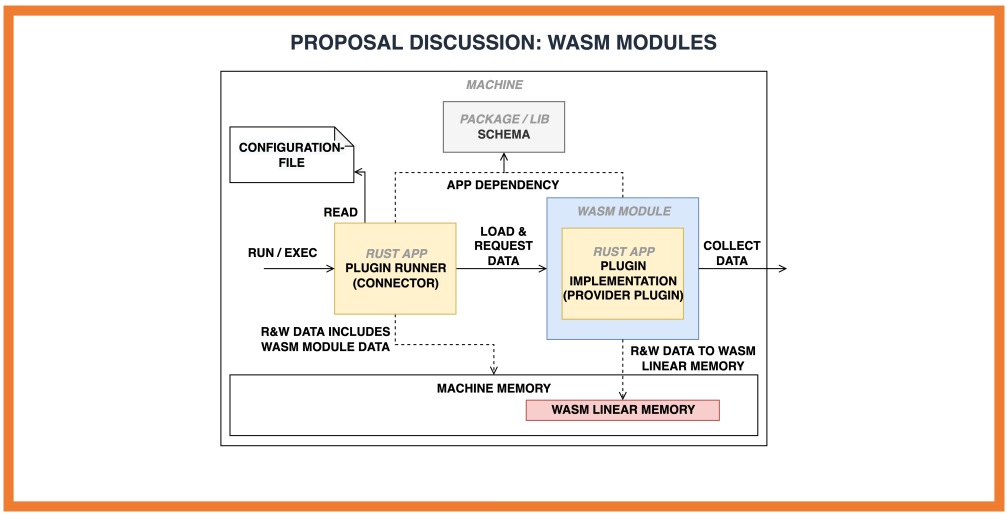

Abbildung 4.13: Carbonaut Provider-Plugins: Diskussionsgrundlage WASM Module

Die Anbindung von WASM-Modulen wird hauptsächlich von der Bibliothek `wasmtime` übernommen, welche Funktionalitäten zum Starten, Stoppen und Ausführen von Methoden in WASM-Modulen bietet. Zusätzlich ermöglicht sie die Transformation eines WASM-gefilterten Pointers zu einem regulären Pointer. Da WASM-Module bei Methodenaufrufen nur Ganzzahlen zurückgeben können, ist die Anbindung nur durch Umwege möglich. Die Anbindung erfolgt wie folgt:

1. **WASM-Modul starten**: Basierend auf der Konfiguration werden Plugins als WASM-Module gestartet. Die WASM-Binaries werden entweder lokal referenziert oder von einem remote heruntergeladen.

2. **Daten anfordern**: Carbonaut fordert über einen Methodenaufruf Daten vom WASM-Modul an.

3. **Externe Datenanfrage**: Das WASM-Modul fragt Daten von der externen Schnittstelle an und überträgt diese in das Carbonaut-Schema.

[20] WASM Plugin Projekt: https://github.com/leonardpahlke/example-wasm-pointer-plugins

4. **Daten in ein DTO verpacken**: Da nur von WASM Modulen Integer zurückgegeben werden können, müssen Pointer verwendet werden. Wenn jedoch nur ein Pointer zurückgegeben wird, fehlt die Information, wo das Objekt endet. Die gesammelten Daten haben eine variable Länge $d = \|\text{bytes}\|$. Daher wird ein DTO-Objekt erstellt, das eine feste Länge hat und neben dem Pointer die Länge der gesammelten Daten hält. Der Pointer zum DTO wird an Carbonaut zurückgegeben. Das DTO-Objekt sieht folgendermaßen aus:

   Listing 4.8: DTO Struktur

   ```
   #[repr(C)]
   #[derive(Debug)]
   pub struct PluginCollectData {
       pub offset: i32,
       pub len: i32,
   }
   ```

5. **Pointer-Transformation**: Carbonaut erhält den DTO-Pointer vom WASM-Modul und überträgt diesen von einem WASM-Pointer in einen regulären Pointer. WASM-Module verwalten ihren eigenen linearen Adressraum, der jedoch logischerweise mit der Host-Maschine zusammenhängt.

6. **DTO auspacken**: Carbonaut liest das DTO-Objekt ein und erhält somit den WASM-Modul-Pointer, welcher die Datenverweise und die Länge der Daten enthält, die eingelesen werden müssen. Der eingelesene WASM-Pointer wird ebenfalls übertragen.

7. **Gesammelte Daten einlesen**: Carbonaut liest die vom WASM-Modul gesammelten Daten ein und verarbeitet diese weiter.

8. **Freigeben von Daten im WASM-Speicher**: Carbonaut führt eine weitere Methode aus, um die gespeicherten Daten im WASM-Modul freizugeben (Dealloc-Methode).

Dieser Ansatz hat einige Vorteile:

- Kleine Binaries, da WASM verwendet wird. In dem Beispielprojekt hatte das Provider Plugin WASM Modul 76 KB.

- Anbindung über WASM Module ist performant und hat kaum Overhead. Die wasmtime Bibliothek hat eine Größe von 332K (v19.0.1), was bei einem Projekt bei Kompilierung zusätzlich reduziert wird, wenn nicht alle Funktionalitäten verwendet werden.
- Dieser Ansatz ist Programmiersprachen agnostisch.

Dieser Ansatz hat auch einige Nachteile:

- WASM ist eine neue Technologie, die adaptiert wird, und nicht für Stabilität bekannt ist. Unterstützung für WASM ist oft experimentell und es gibt weniger Entwicklungsunterstützende-Ressourcen.
- Die Schritte Objekte von WASM Provider-Plugins zu Carbonaut zu bekommen basiert auf Pointer „Tricks“.

Alles in allem ist dieser Ansatz spannend und eine interessante Alternative zu dem Ansatz rund um Prozessen. Durch eine höhere Komplexität Carbonaut zu verwalten, was von den Kernzielen von Carbonaut wegführt, ist dieser Ansatz in dem initialen Design für Carbonaut nicht erstrebenswert. WASM könnte Basis einer Weiterentwicklung sein, nachdem Interfaces, Schemas und weiteres gereift ist.

4.10 Qualitätsanforderungen

Dieser Abschnitt beinhaltet konkrete Qualitätsszenarien, welche die zentralen Qualitätsziele aus Abschnitt 4.1.2, aber auch andere geforderte Qualitätseigenschaften besser fassen. Sie ermöglichen es, Entscheidungsoptionen zu bewerten.

4.10.1 Qualitätsbaum

Die folgende Abbildung 4.14 gibt in Form eines sogenannten Qualitätsbaumes (*Utility Tree*) einen Überblick über die relevanten Qualitätsmerkmale und ordnet ihnen Szenarien als Beispiele zu. Die Qualitätsszenarien werden im folgenden Abschnitt 4.10.2 besprochen. Die Qualitätsziele, aus Abschnitt 4.1.2, sind in der Abbildung ebenfalls enthalten und verweisen jeweils auf die Szenarien, welche sie illustrieren.

Qualitätskriterien orientieren sich an den ISO 25019 Qualitätskriterien, die in Abschnitt 3.1.4 besprochen wurden. Nachhaltigkeit wurde als zusätzliches Qualitätskriterium aufgenommen. Punkte unter Nachhaltigkeit könnten auch unter Wartbarkeit, Effizienz und Übertragbarkeit verstreut werden. Die Bildung eines eigenen Abschnittes jedoch hebt die Relevanz hervor und clustert alle Nachhaltigkeitsthemen unter einem Block. Es wurden die folgenden Punkte unter Nachhaltigkeit definiert.

- **Energieeffizienz**: Maßnahmen zur Minimierung des Energieverbrauchs sowohl bei der Entwicklung als auch beim Einsatz der Software.
- **Langlebigkeit**: Design-Prinzipien, die sicherstellen, dass die Software über lange Zeiträume hinweg nützlich bleibt und leicht aktualisiert werden kann.
- **Materialeffizienz**: Minimierung des Hardwareverbrauchs durch optimierte Software, die weniger leistungsfähige Hardware benötigt.

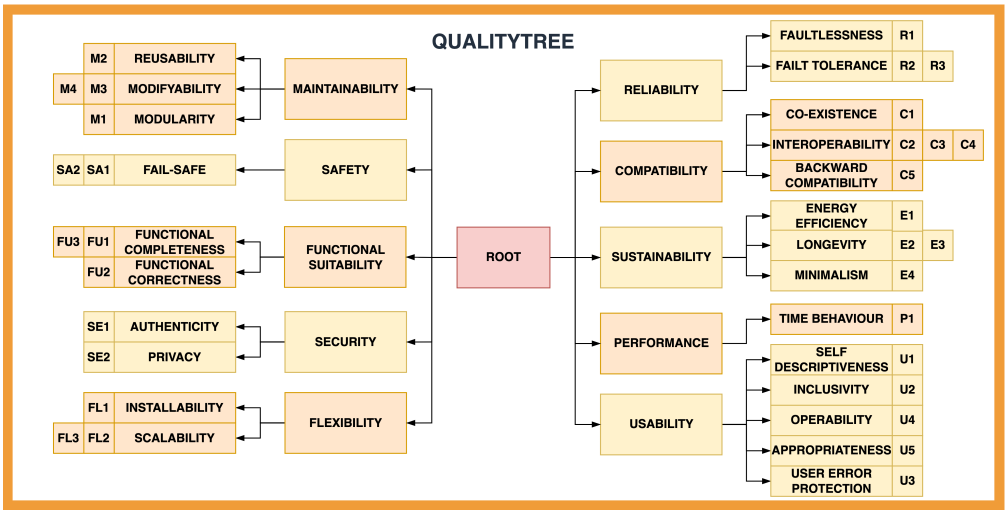

Abbildung 4.14: Qualitätsbaum

4.10.2 Qualitätsszenarien

Die Anfangsbuchstaben der Bezeichner (IDs) der Szenarien in der folgenden Tabelle 4.9 stehen jeweils für das übergeordnete Qualitätsmerkmal, `W` beispielsweise für Wartbarkeit. Diese Bezeichner finden auch im Qualitätsbaum Verwendung, welcher in dem Abschnitt 4.10.1 vorgestellt wurde. Nicht immer lassen sich die Szenarien eindeutig einem Merkmal zuordnen. Sie treten daher mitunter mehrmals im Qualitätsbaum auf.

ID	Szenario
M1	Provider-Plug-ins sind modular
M2	Das System fügt sich in das Cloud Native Ökosystem ein.
M3	Provider-Plugins können von Nutzern hinzugefügt werden.
M4	Das System ist als Open-Source-Projekt öffentlich und weiterentwickelbar.
SA1	Das System kehrt bei verschwundener oder abgestürzter Infrastruktur auf den Basiszustand zurück.
SA2	Das System korrigiert einen fehlerhaften internen state automatisch.
FL1	Das System besteht aus einer Binary, die flexibel eingesetzt werden kann.
FL2	Das System geht davon aus, dass sich die observierte Infrastruktur zur Laufzeit wandelt (vertikale Skalierbarkeit).
FL3	Das System kann mehrfach eingesetzt werden (horizontale Skalierbarkeit).
SE1	Authentizität wird außerhalb des Systems konfiguriert und validiert.
SE2	Das Projekt verarbeitet keine personenbezogenen Daten.
R1	Bei unvollständigen Daten wird ein Log erstellt und dann weiterverfahren.
R2	Fehlerhafte Konfiguration wird nicht verarbeitet.
R3	Fehlerhafte Provider-Plugins führen nicht zum Versagen des restlichen Systems.
U1	Nutzer können den internen Zustand von Carbonaut abfragen.
U2	Das Projekt ist in Englisch entwickelt, was eine breitere Öffentlichkeit versteht.
U3	Nicht genutzte Konfigurationseinstellungen wird dem Nutzer sichtbar gemacht.
U4	Das System kann als Binary oder Container ausgeführt werden.
U5	Das Projekt ist auf einer Dokumentationswebsite beschrieben.
C1	Carbonaut verändert andere Systeme nicht (nur `READS`, keine `WRITES`).
C2	Provider-Interfaces und Datenschema bilden die Interoperabilitätsebene.
C3	Container-Images werden für alle gängigen Plattformen erstellt und verteilt.
C4	Das System veröffentlicht Daten in einem generischen Schema.
C5	Das System ist abwärtskompatibel sein („backward compatibility“) und stellt sicher, dass ältere Versionen weiterhin unterstützt werden.
P1	Werden unvollständige Daten verarbeitet, so wartet Carbonaut nicht auf die restlichen Daten, sondern springt zum nächsten Schritt, um die Verarbeitung nicht zu stoppen.
FU1	Carbonaut veröffentlicht keine in Teilen unvollständigen Daten.
FU2	Carbonaut sammelt immer alle Daten angebundener Systeme.
FU3	Das System sollte alle notwendigen Funktionen bieten, um den vollständigen Anwendungsfall abzudecken, für den es entwickelt wurde.
E1	Das System fährt bei nicht Benutzung auf ein Minimum herunter (Idle State).
E2	Das System versucht so wenige Abhängigkeiten zu anderen Projekten und Bibliotheken aufzubauen. Wenn Abhängigkeiten entstehen, dann zu etablierten Projekten, die eine lange Laufzeit generieren können.
E3	Erstellte Schnittstellen und Schemata werden ab dem Release v1 fortführend unterstützt.
E4	Das System beinhaltet nur Funktionalität, dass das Kernziel von Carbonaut dient.

Tabelle 4.9: Carbonaut Qualitätsszenarien

4.11 Risiken und technische Schulden

Die folgenden Risiken wurden zu Beginn des Vorhabens als Probleme, die auftreten können, identifiziert. Sie beeinflussten die Planung maßgeblich. Seit Abschluss der Implementation Anfang Juni 2024 gelten sie als größtenteils beherrscht und eingeordnet.

4.11.1 Datenintegration und Erstellung höherwertiger Daten

***Status**: Risiko bewältigt*

Eine Kernfunktionalität von Carbonaut ist das Zusammenführen und Verbinden gesammelter Daten. Carbonaut erstellt höherwertige Daten, da es durch das Zusammenführen verschiedener Daten Aussagen treffen kann, die ohne Hinzuziehen weiterer Informationen nicht möglich wären. Nachhaltigkeit ist ein aufkommendes Feld in der Informatik, das mit Unsicherheiten und Experimenten einhergeht. Carbonaut könnte als ein solches Experiment aufgefasst werden. Da sich dieses Feld noch entwickelt, werden sich die Schnittstellen von IT-Infrastruktur-Anbietern anpassen und könnten in Zukunft weitere Informationen veröffentlichen, was das Carbonaut-Projekt beeinflussen würde. Ein Grund, warum so viele unterschiedliche Datenquellen zusammengeführt werden, ist das Fehlen nachhaltigkeitsorientierter APIs. Neben den Schnittstellen befinden sich auch rechtliche Verordnungen in der Entwicklung und könnten zusätzliche Auflagen in Bezug auf die Nachhaltigkeit von IT-Systemen und Softwarebetreibern erlassen.

Eventualfallplanung

Da die Implementierungen der Provider-Plugins in Carbonaut direkt erfolgen, werden Änderungen an APIs, Provider-Interfaces und Datenschemata sofort propagiert und können zeitnah angepasst werden.

Risikominderung

Subsysteme im Projekt sind voneinander entkoppelt. Das Provider-Plugin-Subsystem ist zudem modular aufgebaut und kann flexibel erweitert werden. Durch die geringe Komplexität der Plugins wird die weitere Adaption erleichtert. Diese modulare Struktur

ermöglicht es, neue Anbieter schnell zu integrieren und das System kontinuierlich zu verbessern, ohne den gesamten Betrieb zu gefährden.

4.11.2 Relevanz

***Status**: Risiko minimiert, aber bisher nicht bewältigt*

Das Projekt lebt durch die Anbindung und Integration verschiedener Datenanbieter. Die Qualität und der Nutzen des Projekts wachsen mit der Anzahl der Nutzer. Wenn mehr Nutzer das System mit ihren spezifischen Anforderungen und IT-Infrastrukturen einsetzen und anpassen, steigt die Relevanz des Projekts. Dies gilt zwar für jedes Open-Source-Projekt, jedoch hat es bei Carbonaut eine besondere Bedeutung. Die Relevanz und Qualität des Projekts hängen somit eng mit seiner Adaption und Verbreitung zusammen.

Eventualfallplanung

Im Laufe der Zeit werden weitere Plattformen hinzugefügt, um die Vielfalt des Projekts zu erhöhen.

Risikominderung

Das Projekt ermöglicht, neue Provider-Plugins hinzuzufügen, wenn diese bisher nicht existieren. Der Prozess zur Integration neuer Plugins ist dokumentiert und beschrieben. Die Komplexität diesen Aufwand zu erbringen ist überschaubar und setzt kein weiteres Wissen über den Rest von Carbonaut voraus. Dadurch wird die Einstiegshürde für neue Nutzer verringert, die das Projekt interessant finden und es erweitern möchten.

Natürlich, hier ist der korrigierte und erweiterte Text:

4.11.3 Provider-Plugin Anbindung

***Status**: Das Risiko ist beherrscht. Es wurden auch Strategien in Abschnitt 4.9.2 für die weitere Entwicklung des Projekts entwickelt, sollte das Risiko an Bedeutung gewinnen.*

Das Anbinden von externen Datenquellen über das Provider-Plugin-Subsystem ist eine der Kernfunktionalitäten von Carbonaut. Diese Anbindung ist nicht trivial und erfordert sorgfältige Planung und Implementierung. Werden hier schlechte Designentscheidungen getroffen, so wird das Problem mit der Anzahl an Plugins, Nutzern des Projektes und open source Entwickler multipliziert.

Eventualfallplanung

Es werden Strategien entwickelt, um das Projekt weiterzuentwickeln, wenn es in Bezug auf die Anzahl der Plugins, die Anzahl der Nutzer oder die Anzahl der Open-Source-Beitragenden wächst. Dazu gehört die kontinuierliche Überwachung der Systemleistung mithilfe von Performance-Tests.

Risikominderung

Es wurden verschiedene Strategien zur Implementierung des Provider-Plugin-Subsystems erarbeitet und abgewogen. Diese Ansätze werden in Abschnitt 4.9.2 erörtert.

4.12 Glossar

Das folgende Glossar in der Tabelle 4.10 erklärt verwendete Fachbegriffe. Dieses Glossar enthält nur Fachbegriffe und keine allgemeinen Begriffe der Softwareentwicklung, wie API, Schnittstelle, Datenschema oder Ähnliches.

Fachbegriff	Beschreibung
IT-Infrastruktur	Unter IT-Infrastruktur verstehen wir die Gesamtheit der technischen Komponenten, die zur Bereitstellung und Verwaltung von IT-Diensten notwendig sind. IT-Infrastruktur-Anbieter bieten Schnittstellen, um IT-Ressourcen zu verwalten. Carbonaut greift auf diese Schnittstellen zurück und bindet diese über das Provider-Plugins-Subsystem an.
IT-Ressource	Eine IT-Ressource bezeichnet alle digitalen Mittel, wie Rechenleistung, Speicher und Netzwerkkapazitäten, die von IT-Infrastruktur-Anbietern bereitgestellt werden. Carbonaut greift auf IT-Ressourcen von konfigurierten IT-Infrastruktur-Anbietern zurück, um Daten zu sammeln.
Go	Go ist eine Programmiersprache, die sich in der Cloud-Native-Community durch ihre Effizienz, ihren Minimalismus und ihre Gleichzeitigkeit verbreitet hat. Carbonaut nutzt Go für die Implementierung seiner Kernfunktionalitäten.
GitHub	GitHub ist eine Plattform zur Versionsverwaltung und gemeinsamen Entwicklung von Softwareprojekten. GitHub stellt die Open-Source-Plattform für das Carbonaut-Projekt[21] sowie die Carbonaut-Dokumentation[22] bereit.
Kubernetes	Kubernetes ist eine Open-Source-Plattform zur Automatisierung der Bereitstellung, Skalierung und Verwaltung von containerisierten Anwendungen. Carbonaut kann in einem Kubernetes-Cluster verwaltet werden. Dieser Ansatz ist Schritt für Schritt in der Dokumentation[23] beschrieben und als Verteilungssicht in Abschnitt 4.7.1 beschreiben.
gRPC	gRPC ist ein modernes, leistungsfähiges Framework zur Remote Procedure Call (RPC)-Kommunikation. gRPC findet bei Carbonaut derzeit keine Verwendung, ist aber Teil der Architekturdiskussion in Abschnitt 4.9.2 und könnte in Zukunft Basis einer Weiterentwicklung von Carbonaut sein.
WASM	WebAssembly (WASM) ist ein binäres Anweisungsformat, das es ermöglicht, Code auf verschiedenen Plattformen mit nahezu nativer Geschwindigkeit auszuführen. WASM findet bei Carbonaut derzeit keine Verwendung, ist aber Teil der Architekturdiskussion in Abschnitt 4.9.2 und könnte in Zukunft Basis einer Weiterentwicklung von Carbonaut sein.

Tabelle 4.10: Carbonaut Glossar

5 Verwendung von Carbonaut - Szenarien

In diesem Kapitel wird das im vorherigen Kapitel 4 vorgestellte Projekt Carbonaut in verschiedenen Szenarien eingesetzt. Ein Szenario kann auch als Use Case betrachtet werden. Ein Szenario dient dazu, die zuvor in Abschnitt 4.1 und 4.10 festgelegten Qualitätsziele anschließend zu validieren und deren Implementierung zu diskutieren. Die Validierung und Diskussion findet im kommenden Kapitel 6 statt. In diesem Kapitel wird dafür die Datengrundlage gegeben.

Es werden zwei Szenarien durchlaufen, die sich an der Laufzeitsicht, aus Abschnitt 4.6, und an der Verteilungssicht, aus Abschnitt 4.7, orientieren. Diese Szenarien werden nun beschrieben, durchgeführt und Ergebnisse beobachtet. Dieses Kapitel beginnt mit einer Einführung, wie Szenarien allgemein umgesetzt werden.

5.1 Szenario Einführung

Bevor die Szenarien durchgeführt werden, wird betrachtet, wie Szenarien aufgebaut sind und wie diese umgesetzt werden. Der Prozessablauf eines Szenarios mit Durchführung wird anschließend beschrieben.

Da Carbonaut auf eine IT-Infrastruktur zurückgreift, um Daten zusammenzutragen, ist die Durchführung von Szenarien komplexer, als Carbonaut nur in unterschiedlichen Konfigurationen zu starten. Das Verwalten und Konfigurieren der IT-Infrastruktur sind Teil jedes Szenarios. Im groben definiert ein Szenario die folgenden vier Aspekte, um durchgeführt werden zu können.

1. **Szenarioprozesses Definition**: wie sieht die IT-Infrastruktur auf, die Carbonaut betrachtet und wie verändert sich die Infrastruktur über die Zeitspanne des Szenarios?

2. **IT-Ressourcen Konfiguration**: welche Eigenschaften haben verwaltete IT-Ressourcen, wenn diese erstellt werden?
3. **Szenarioprozesses Definition**: wie sieht die Laufzeitsicht des Szenarios aus? Wann werden Messungen durchgeführt, wann wird die IT-Infrastruktur angepasst?
4. **Bereitstellung von Carbonaut**: wie wird Carbonaut verwaltet und konfiguriert.

Für den ersten wie auch den zweiten Punk der Liste können, Szenario-unabhängige Strukturen entwickelt werden, die sowie im ersten Szenario als auch im zweiten Szenario eingesetzt werden. In den folgenden beiden Abschnitten, 5.1.1 und 5.1.2, wird dies besprochen.

5.1.1 Verwaltung der IT-Infrastruktur

Die Szenarien verwenden Terraform zur Beschreibung, Konfiguration und Verwaltung der IT-Infrastruktur. Terraform ist ein Open-Source-Projekt, das Plugins zur Implementierung und Konfiguration von Schnittstellen von IT-Infrastrukturanbietern verwendet. Diese Plugins ermöglichen die Integration mit verschiedenen Anbietern und Diensten, um eine konsistente Infrastrukturverwaltung zu gewährleisten.

Die IT-Infrastruktur-Topologie wird über `.tf`-Dateien beschrieben. Diese Dateien enthalten die Definitionen der Ressourcen und deren Konfigurationen. Über eine CLI (Command Line Interface) kann diese IT-Infrastruktur erstellt und gelöscht werden. Variablen ermöglichen es, Parameter wie die Anzahl der erstellten Ressourcen anzupassen.

Im nachfolgenden Beispiel ist ein Auszug der verwendeten Terraform-Konfiguration gezeigt.

Listing 5.1: Auszug - Terraform Definition Carbonaut Szenarien

```
resource "equinix_metal_project" "dev_project" {
  name = "carbonaut-dev"
}

resource "equinix_metal_project_ssh_key" "dev_key" {
  name       = "dev"
  public_key = var.public_key
  project_id = equinix_metal_project.dev_project.id
```

```
}

resource "equinix_metal_device" "development_server" {
  hostname         = "tf.coreos2"
  plan             = "c3.small.x86"
  metro            = "fr"
  operating_system = "debian_12"
  project_ssh_key_ids = [equinix_metal_project_ssh_key.dev_key.id]
  billing_cycle    = "hourly"
  project_id       = equinix_metal_project.dev_project.id
}
```

In diesem Beispiel werden folgende Ressourcen erstellt:

- Ein Projekt namens `carbonaut-dev` im Equinix Metal.
- Ein SSH-Schlüssel namens `dev` für den Zugriff auf das Projekt.
- Ein Entwicklungsserver mit dem Hostnamen `tf.coreos2`, der den Plan `c3.small.x86` verwendet und im Metro-Bereich `fr` (Frankfurt) bereitgestellt wird. Der Server läuft auf dem Betriebssystem Debian 12 und verwendet den zuvor definierten SSH-Schlüssel zur Authentifizierung.

Die eingesetzte Terraform Konfiguration ist auf GitHub[1] einsehbar.

5.1.2 Konfiguration von erstellten IT-Ressourcen

Die Szenarien nutzen Ansible Playbooks, um erstellte IT-Ressourcen zu konfigurieren. Ansible Playbooks definieren Schritte, die ausgeführt werden müssen, um beispielsweise alle nötigen Libraries auf der Maschine zu installieren. In einem Manifest werden alle Ressourcen Schnittstellen referenziert, die den erwünschten State haben sollen.

[1] Carbonaut/dev: https://github.com/leonardpahlke/carbonaut/tree/main/dev

5.1.3 Stress Tests

Die Szenarien führen Stresstests auf den erstellten Ressourcen durch, damit Laufzeitdaten generiert werden. Der Stresstest ist in einem Bash-Skript beschrieben, welches über ein Ansible-Playbook auf die Ressource kopiert und ausgeführt wird. Der Stresstest nutzt `stress`, `memtester` und `fio`, um verschiedene Komponenten der IT-Ressource zu belasten. Im nachfolgenden Codeausschnitt ist ein Teil des Stresstests gezeigt.

Listing 5.2: Aussschitt des Stress Test Bash Skripts

```
run_cpu_stress_test() {
    echo "Running CPU stress test with ${CPU_WORKERS} workers for \
        ${CPU_TIMEOUT}..."
    stress --cpu "${CPU_WORKERS}" --timeout "${CPU_TIMEOUT}"
}

run_memory_stress_test() {
    echo "Running memory stress test with ${MEMORY_SIZE}MB for \
        ${MEMORY_ITERATIONS} iterations..."
    memtester "${MEMORY_SIZE}" "${MEMORY_ITERATIONS}"
}
```

5.2 Carbonaut Szenario – Laufzeitsicht

Dieses Szenario schließt an der Carbonaut Laufzeitsicht an, die in Abschnitt 4.6 vorgestellt wurde.

5.2.1 Szenarioprozesses Definition

Tabelle 5.1 zeigt den Szenarioablauf. Pro Schritt werden Metrikdaten, der interne Status von Carbonaut und alle erzeugten Protokolle gespeichert. Die Spalte *Nr.* gibt den Prozessschritt an, während die Spalte *Beschreibung & Ziel* den Schritt und den Grund für dessen Ausführung erklärt. In der Spalte *Infrastruktur* wird angegeben, welche Objekte im Zustand nach Anpassung der Infrastruktur gefunden werden sollen. Die Spalte *Plugins* beschreibt, welche Provider-Plugins in dieser Phase verwendet werden.

Nr.	Beschreibung & Ziel	Infrastruktur	Plugins
1	**Initialisierung**; Validierung, ob die Infrastruktur abgebildet wird und Daten gesammelt werden können.	$\{a_1\} = \lvert\text{Account}\rvert$ $\{p_1\} = \lvert\text{Projekt}_{a_1}\rvert$ $\{r_1, r_2\} = \lvert\text{Ressource}_{a_1 p_1}\rvert$	Equinix Elec. Map Scaphandre
2	**Dereferenzierung**; Validierung, ob der angelegte State zurückgesetzt wird.	$\varnothing$	$\varnothing$
3	**Wiederzuweisung**; Validierung, ob ein Account zur Laufzeit wieder angelegt wird.	$\{a_1\} = \lvert\text{Account}\rvert$ $\{p_1\} = \lvert\text{Projekt}_{a_1}\rvert$ $\{r_1, r_2\} = \lvert\text{Ressource}_{a_1 p_1}\rvert$	Equinix Elec. Map Scaphandre
4	**Ressourcen Erstellung**; Validierung, ob eine neu erstellte Ressource nachgepflegt wird.	$\{a_1, a_2\} = \lvert\text{Account}\rvert$ $\{p_1\} = \lvert\text{Projekt}_{a_1}\rvert$ $\{r_{1a}, r_{2a}\} = \lvert\text{Ressource}_{a_1 p_1}\rvert$	Equinix Elec. Map Scaphandre
5	**Löschen einer Ressource**; Validierung, ob der State nachgepflegt wird.	$\{a_1\} = \lvert\text{Account}\rvert$ $\{p_1\} = \lvert\text{Projekt}_{a_1}\rvert$	Equinix
6	**Clean Up**; Löschen erstellter Ressourcen.	$\varnothing$	$\varnothing$

Tabelle 5.1: Szenarioprozessschritte 1 – Carbonaut Laufzeitsicht

In diesem Szenario wird Carbonaut als Binary ausgeführt, da der Fokus auf die Funktionale Korrektheit von Carbonaut liegt. Im zweiten Szenario wird die Bereitstellung über Kubernetes betrachtet.

In diesem Szenario wird Carbonaut als Binary ausgeführt, da der Schwerpunkt auf Laufzeitsicht, also der Funktionalität von Carbonaut liegt. Im zweiten Szenario wird die Bereitstellung von Carbonaut über Kubernetes durchgeführt.

5.2.2 Szenario Durchführung

Das erste Szenario besteht aus fünf Schritten, wo Carbonaut getestet wird. In einem sechsten Schritt werden erstellte Ressourcen gelöscht. Die Schritte wurden folgendermaßen durchgeführt. Jeder Schritt hat eine ID, welche die Identifikation der Schritte in den erstellten Dateien vereinfacht. `s1-s1-1` steht für Szenario 1, Schritt 1, Befehl 1.

S1: Schritt Nr. 1:

1. **[s1-s1-1]**: Erstellung der Equinix-Infrastruktur mit 1x Projekt und 1x Ressource (siehe Abschnitt 5.1.1).
2. **[s1-s1-2]**: Konfigurierung der erstellten Ressource (siehe Abschnitt 5.1.2).
3. **[s1-s1-3]**: Verifizierung, dass erstellte Ressourcen konfiguriert und verfügbar sind.
4. **[s1-s1-4]**: Durchführung von Stresstests zur Etablierung einer Datengrundlage (siehe Abschnitt 5.1.3).
5. **[s1-s1-5]**: Ausführen von Carbonaut mit einer Konfiguration, die auf die erstellte IT-Infrastruktur verweist.
6. **[s1-s1-6]**: Zugriff und Speicherung des Carbonaut-State zur Validierung.
7. **[s1-s1-7]**: Zugriff und Speicherung der Carbonaut-Metriken zur Validierung.

S1: Schritt Nr. 2:

1. **[s1-s2-1]**: Update der Carbonaut Konfiguration, mit einer leeren Konfiguration, die auf keine IT-Infrastruktur verweist.
2. **[s1-s2-2]**: Zugriff und Speicherung des Carbonaut-State zur Validierung.
3. **[s1-s2-3]**: Zugriff und Speicherung der Carbonaut-Metriken zur Validierung.

S1: Schritt Nr. 3:

1. **[s1-s3-1]**: Update der Carbonaut Konfiguration, dass die IT-Infrastruktur wieder gefunden wird.
2. **[s1-s3-2]**: Zugriff und Speicherung des Carbonaut-State zur Validierung.
3. **[s1-s3-3]**: Zugriff und Speicherung der Carbonaut-Metriken zur Validierung.

S1: Schritt Nr. 4:

1. **[s1-s4-1]**: Erstellung einer weiteren Equinix-Ressource mit nun 1x Projekt und 2x Ressource (siehe Abschnitt 5.1.1).
2. **[s1-s4-2]**: Konfigurierung der erstellten Ressource (siehe Abschnitt 5.1.2).
3. **[s1-s4-3]**: Verifizierung, dass erstellte Ressourcen konfiguriert und verfügbar sind.

4. **[s1-s4-4]**: Durchführung von Stresstests zur Etablierung einer Datengrundlage (siehe Abschnitt 5.1.3).
5. **[s1-s4-5]**: Zugriff und Speicherung des Carbonaut-State zur Validierung.
6. **[s1-s4-6]**: Zugriff und Speicherung der Carbonaut-Metriken zur Validierung.

S1: Schritt Nr. 5:

1. **[s1-s5-1]**: Erstellung der Equinix-Infrastruktur mit 1x Projekt und 1x Ressource (siehe Abschnitt 5.1.1).
2. **[s1-s5-2]**: Zugriff und Speicherung des Carbonaut-State zur Validierung.
3. **[s1-s5-3]**: Zugriff und Speicherung der Carbonaut-Metriken zur Validierung.

Nach diesen Schritten werden die erstellten Ressourcen wieder gelöscht, um den ursprünglichen Zustand des Systems wiederherzustellen.

S1: Schritt Nr. 6:

1. **[s1-s6-1]**: Stoppen von Carbonaut
2. **[s1-s6-2]**: Löschen erstellter Equinix Infrastruktur.

Die Ergebnisse sind in dem GitHub Repository[2] unter `/test-scenario/results-1` einsehbar. Ein Beispiel zu dem erhobenen State ist im Anhang unter A.2 gegeben. Ein Beispiel zu den erhobenen Metriken ist im Anhang unter A.3 gegeben. Das Szenario konnte fehlerfrei durchgeführt werden.

5.3 Carbonaut Szenario – Verteilungssicht

Dieses Szenario schließt an der Carbonaut Verteilungssicht an, die in Abschnitt 4.7 vorgestellt wurde.

[2]Projekt Repository: https://github.com/leonardpahlke/carbonaut/

Nr.	Beschreibung & Ziel	Infrastruktur	Plugins
1	**Initialisierung**; Validierung, ob die Infrastruktur abgebildet wird, Carbonaut in einem Kubernetes Cluster eingesetzt werden kann und Daten gesammelt werden können.	$\{a_1\} = \lvert\text{Account}\rvert$ $\{p_1\} = \lvert\text{Projekt}_{a_1}\rvert$ $\{r_1, r_2\} = \lvert\text{Ressource}_{a_1 p_1}\rvert$	Equinix Elec. Map Scaphandre
2	**Clean Up**; Löschen erstellter Ressourcen.	∅	∅

Tabelle 5.2: Szenarioprozessschritte 2 – Carbonaut Verteilungssicht

5.3.1 Szenarioprozesses Definition

Tabelle 5.2 zeigt den Szenarioablauf. Pro Schritt werden Metrikdaten, der interne Status von Carbonaut und alle erzeugten Protokolle gespeichert. Die Spalte *Nr.* gibt den Prozessschritt an, während die Spalte *Beschreibung & Ziel* den Schritt und den Grund für dessen Ausführung erklärt. In der Spalte *Infrastruktur* wird angegeben, welche Objekte im Zustand nach Anpassung der Infrastruktur gefunden werden sollen. Die Spalte *Plugins* beschreibt, welche Provider-Plugins in dieser Phase verwendet werden.

Da in diesem Szenario der Fokus auf die Bereitstellung von Carbonaut liegt, während im vorherigen Szenario die Funktionalität im Vordergrund stand, sind im Prozess lediglich zwei Schritte erforderlich.

In diesem Szenario wird Carbonaut in einem Kubernetes Cluster verwaltet. Die Verteilungssicht wurde im vorherigen Abschnitt 4.7.1 gezeigt. Weitere Schritte in diesem Szenario würden das zuvor durchgeführte Szenario duplizieren.

5.3.2 Szenario Durchführung

Das zweite Szenario besteht aus einem Schritt, wo Carbonaut getestet wird. In einem zweiten Schritt werden erstellte Ressourcen gelöscht. Die Schritte wurden folgendermaßen durchgeführt. Jeder Schritt hat eine ID, welche die Identifikation der Schritte in den erstellten Dateien vereinfacht. `s2-s1-1` steht für Szenario 2, Schritt 1, Befehl 1.

S2: Schritt Nr. 1:

1. **[s2-s1-1]**: Kubernetes Cluster lokal starten. Hier wurde Minikube[3] verwendet.

[3]Minikube: https://minikube.sigs.k8s.io

2. **[s2-s1-2]**: Erstellung der Equinix-Infrastruktur mit 1x Projekt und 1x Ressource (siehe Abschnitt 5.1.1).
3. **[s2-s1-3]**: Konfigurierung der erstellten Ressource (siehe Abschnitt 5.1.2).
4. **[s2-s1-4]**: Verifizierung, dass erstellte Ressourcen konfiguriert und verfügbar sind.
5. **[s2-s1-5]**: Durchführung von Stresstests zur Etablierung einer Datengrundlage (siehe Abschnitt 5.1.3).
6. **[s2-s1-6]**: Erstellen von Kubernetes Ressourcen, die Carbonaut deployen (siehe Abschnitt 4.7.1). Die Kubernetes-Manifestdateien sind im Projekt-Repository unter `dev/k8s.yaml` verfügbar.
7. **[s2-s1-7]**: Kubernetes Port-Forward, damit auf Carbonaut zugegriffen werden kann.
8. **[s2-s1-8]**: Zugriff und Speicherung des Carbonaut-State zur Validierung.
9. **[s2-s1-9]**: Zugriff und Speicherung der Carbonaut-Metriken zur Validierung.

Nach diesem Schritt werden die erstellten Ressourcen wieder gelöscht, um den ursprünglichen Zustand des Systems wiederherzustellen.

S2: Schritt Nr. 2:

1. **[s2-s2-1]**: Lösche erstellte Kubernetes Ressourcen.
2. **[s2-s2-2]**: Löschen erstellter Equinix Infrastruktur.
3. **[s2-s2-3]**: Herunterfahren des lokalen Kubernetes Clusters.

Die Ergebnisse sind in dem GitHub Repository[4] unter `/test-scenario/results-2` einsehbar. Das Szenario konnte fehlerfrei durchgeführt werden.

[4]Projekt Repository: https://github.com/leonardpahlke/carbonaut/

6 Bewertung des Projekts Carbonaut basierend auf Szenarien

In diesem Abschnitt wird das Carbonaut Projekt bewertet. Dies geschieht auf der Grundlage der Architekturdokumentation aus Abschnitt 4, der Implementation, die in dem GitHub Repository `leonardpahlke/carbonaut`[1] vorliegt, und der durchgeführten Testszenarien, die in Abschnitt 5 durchgeführt wurden.

Das Ziel der Bewertung ist festzustellen, ob die definierten Ziele erreicht wurden, die generelle Güte des Projekts zu diskutieren sowie Erkenntnisse aus der Entwicklung zu gewinnen und zu ermitteln, wo weitere Arbeiten anknüpfen könnten.

6.1 Analyse der Testszenarien

In diesem Abschnitt werden die durchgeführten Szenarien aus Abschnitt 5 betrachtet.

6.1.1 Analyse der Ergebnisse

Die Ergebnisse sind in dem Projektordner unter `test-scenario/results-1` und unter `test-scenario/results-2` gespeichert. Das Szenario kann über die ausführung der `test-scenario/scenario-1.bash` und `test-scenario/scenario-2.bash` ausgeführt werden. Bei der Ausführung werden Dateien, die Ausgaben der einzelnen Schritte dokumentieren. Es kann für jeden dokumentierten Schritt im Szenario eine Datei gefunden werden. Die `test-scenario/results-1/s1-log.txt` und `test-scenario/results-2/s2-log.txt` zeigt den gesamten Ablauf. Im Anhang ist ein Beispiel des Carbonaut-State A.2 und der gesammelten Carbonaut-Metriken unter 5.2.1 gegeben

[1] Projekt Repository: https://github.com/leonardpahlke/carbonaut/

- In dieser Datei finden sich zwischendurch zusätzliche Schritte `[s1-sx-x] updating known_hosts SSH for 145.40.94.191`, die nicht vorab definiert wurden, das Szenario auch nicht weiter beeinflussen, sondern eine Konsequenz ist, wie SSH externe IPs verifiziert. Neue IPs müssen in der `known_hosts`-Datei abgelegt sein.

- Teilweise fehlen Dateien für einzelne Unterschritte. Der Output dieser Schritte ist in diesen Fällen unter `test-scenario/results-1/s1-log.txt` oder `test-scenario/results-2/s2-log.txt` zu finden und umfasst oft nur eine Zeile, weshalb auf eine weitere Datei verzichtet wurde.

- In dem zweiten Szenario wurden zwei Warnungen zurückgegeben, die zwei Python Interpreter aufzeigt. Diese Warnung hat keinen weiteren Einfluss auf das Szenario gehabt, sollte jedoch zeitnah behoben werden.

Listing 6.1: Python Warnung zur Konfiguration der IT-Ressourcen

```
[WARNING]: Platform linux on host 145.40.94.229 is using the
discovered Python interpreter at /usr/bin/python3.11, but future
installation of another Python interpreter could change
the meaning of that path.
```

- Die Carbonaut State und Metrik Daten stimmen mit der Infrastruktur über ein, jedoch fehlen zwei Dateneinträge unter den dynamischen Ressourcendaten. Dies ist bei allen `-metrics.json` Dateien zu beobachten. Hier ein Beispiel, dass zeigt, dass `energy_host_mirojoules` und `cpu_load_percentage` auf `0` steht. Da `cpu_frequency` durchgängig Werte hat, die sich ändern, ist davon auszugehen, dass zwar die Plugin Implementierung funktioniert, aber nicht alle Daten erhoben werden. Dies ist kein Carbonaut Problem, sondern ein Problem, wie in diesem Fall Scaphandre installiert wird. Da Cloud-Anbieter steuern können, auf welche Schnittstellen virtuelle Maschinen zugreifen können, kann dazu führen, dass Energiedaten nicht propagiert und verfügbar sind. Dies wurde in Abschnitt 3.3.3 besprochen.

Listing 6.2: Ausschnitt Carbonaut Metriken in JSON format

```
{
   "equinix":{                  // Account Referenz (Konfiguration)
      "72e5de9e-XXX":{          // Projekt ID
         "f9f6cbcb-YYY":{       // Ressource ID
            "dynamic_data":{
```

```
                "res_data":{
                    "cpu_frequency":4301,
                    "energy_host_mirojoules":0,
                    "cpu_load_percentage":0
                },
                "env_data":{
                    "solar_percentage":50.13,
                    // ...
                }
            },
            "static_data":{
                // ...
            }
        }
    }
}
```

6.1.2 Relevanz und Abdeckung der Szenarien

Zur Bewertung, wie viele Konfigurations-Möglichkeiten die beiden Szenarien abgebildet haben, muss zunächst analysiert werden, welche Einstellungen variiert werden können. Diese Variationen müssen anschließend miteinander potenziert werden. Danach wird betrachtet, welche Fälle abgedeckt wurden.

Carbonaut Szenario Multiplizierung

Das Carbonaut Deployment kann über die folgenden drei Eigenschaften gesteuert werden:

- K: Anpassung der Konfigurationsdatei.
- I: Anpassung der observierten IT-Infrastruktur.
- D: Anpassung, wie Carbonaut verwaltet wird.

Var	Mögliche Werte	Beschreibung
D_k	$\{0, 1\}$	Carbonaut wird in einem Kubernetes Cluster ausgeführt.
D_b	$\{0, 1\}$	Carbonaut wird als Binary ausgeführt.
I_p	$\{0, 1, > 1\}$	Anzahl der Projekte.
I_{p_r}	$\{0, 1, > 1\}$	Anzahl der Ressourcen pro Projekt (ist 0 wenn $I_p = 0$ ist).
K_a	$\{0, 1, > 1\}$	Anzahl der Accounts.
K_{p1}	$\{a1, b1\}$	Verwendetes Provider Plugin für statische Ressourcendaten (es gibt jedoch aktuell nur eins).
K_{p2}	$\{a2, b2\}$	Verwendetes Provider Plugin für dynamische Ressourcendaten (es gibt jedoch aktuell nur eins).
K_{p3}	$\{a3, b3\}$	Verwendetes Provider Plugin für dynamische Environmentdaten (es gibt jedoch aktuell nur eins).

Tabelle 6.1: Szenario Variationen

Berechnung der Variationen

Damit kommen die folgenden Fälle zustande. Alle Fälle, die $b1$, $b2$ oder $b3$ beinhalten wurden aus der Liste gestrichen, da bei Carbonaut jeweils nur ein Plugin implementiert wurde und daher diese Szenarien nicht abgebildet werden konnten. Für eine breitere Abdeckung wäre mindestens zwei Plugins pro Provider Interface nötig.

- Fall 1: $D_k = 1$, $D_b = 0$, $I_p = 1$, $I_{p_r} = 1$, $K_a = 1$, $K_{p1} = a1$, $K_{p2} = a2$, $K_{p3} = a3$
- Fall 2: $D_k = 1$, $D_b = 0$, $I_p = 0$, $I_{p_r} = 0$, $K_a = 0$, $K_{p1} = a1$, $K_{p2} = a2$, $K_{p3} = a3$
- Fall 3: $D_k = 1$, $D_b = 0$, $I_p = 1$, $I_{p_r} => 1$, $K_a => 1$, $K_{p1} = a1$, $K_{p2} = a2$, $K_{p3} = a3$
- Fall 4: $D_k = 1$, $D_b = 0$, $I_p => 1$, $I_{p_r} = 1$, $K_a = 1$, $K_{p1} = a1$, $K_{p2} = a2$, $K_{p3} = a3$
- Fall 5: $D_k = 1$, $D_b = 0$, $I_p => 1$, $I_{p_r} => 1$, $K_a => 1$, $K_{p1} = a1$, $K_{p2} = a2$, $K_{p3} = a3$
- Fall 6: $D_k = 0$, $D_b = 1$, $I_p = 0$, $I_{p_r} = 0$, $K_a = 0$, $K_{p1} = a1$, $K_{p2} = a2$, $K_{p3} = a3$
- Fall 7: $D_k = 0$, $D_b = 1$, $I_p = 1$, $I_{p_r} = 1$, $K_a = 1$, $K_{p1} = a1$, $K_{p2} = a2$, $K_{p3} = a3$
- Fall 8: $D_k = 0$, $D_b = 1$, $I_p = 1$, $I_{p_r} => 1$, $K_a => 1$, $K_{p1} = a1$, $K_{p2} = a2$, $K_{p3} = a3$
- Fall 9: $D_k = 0$, $D_b = 1$, $I_p => 1$, $I_{p_r} = 1$, $K_a = 1$, $K_{p1} = a1$, $K_{p2} = a2$, $K_{p3} = a3$

Fall Nr.	Implementiert	Beschreibung
Fall 1	+	Implementiert in dem Szenario 2 Schritt 1
Fall 2–5	–	Wurden nicht implementiert. Es wurde argumentiert, dass diese Variationen über die Verwendung von Binaries (Fall 6–10) abgedeckt werden können.
Fall 6	+	Implementiert in Szenario 1 Schritt 2
Fall 7	+	Implementiert in Szenario 1 Schritt 1
Fall 8	–	Nicht implementiert
Fall 9	+	Implementiert in Szenario 1 Schritt 4
Fall 10	–	Nicht implementiert

Tabelle 6.2: Szenario Variationen

- Fall 10: $D_k = 0$, $D_b = 1$, $I_p => 1$, $I_{p_r} => 1$, $K_a => 1$, $K_{p1} = a1$, $K_{p2} = a2$, $K_{p3} = a3$

In der Liste sind die Fälle 1 - 5 Kubernetes Deployments und die Fälle 6 - 10 sind Binary Deployments. In der folgenden Liste wird bewertet, welche Fälle implementiert wurden und welche fehlen.

Bei tieferer Betrachtung des Codes ist zu erkennen, dass nicht implementierten Fällen (abgesehen von 2 bis 5) über Unit-Tests abgedeckt wurden.

6.2 Diskussion der Qualitätsziele

In Abschnitt 4.10.2 der Architekturdokumentation wurden die Qualitätsziele des Projektes definiert, die in diesem Abschnitt besprochen werden.

6.2.1 Bewertung der Zielerreichung

In diesem Abschnitt werden die Qualitätsziele von Carbonaut diskutiert. Anschließend wird in der Tabelle 6.3 eine Übersicht zu allen Qualitätszielen gegeben.

M1: Provider-Plug-ins sind modular

Dieses Ziel wurde erreicht. Es wurden Provider Interfaces definiert, die es ermöglichen, Plugins zu implementieren. Plugins können als Module entweder genutzt werden oder nicht.

M2: Das System fügt sich in das Cloud Native Ökosystem ein

Das Ziel wurde teilweise erreicht. Carbonaut bietet die Möglichkeiten in einem Kubernetes Cluster eingesetzt zu werden, und kann von anderen Plugins angebunden werden. Aktuell werden jedoch Metriken nur in JSON-format veröffentlicht und nicht als OpenTelemetry oder Prometheus Metriken. Dies wäre ein wichtiger Baustein für die Integration des Projektes in einem Cloud-Native-System.

M3: Provider-Plugins können von Nutzern hinzugefügt werden

Das Ziel wurde erreicht. Das Projekt ist öffentlich zugänglich. Es wurden Provider Interfaces definiert, die es ermöglichen, Plugins zu implementieren.

M4: Das System ist als Open-Source-Projekt öffentlich und weiterentwickelbar

Das Ziel wurde erreicht. Das Projekt ist öffentlich zugänglich und unter der Apache-2.0 Lizenz frei verfügbar.

SA1: Das System kehrt bei verschwundener oder abgestürzter Infrastruktur auf den Basiszustand zurück

Das Ziel wurde erreicht. Dies wurde in dem Szenario 1 Fall 6 überprüft. Siehe Abschnitt 6.1.1.

SA2: Das System korrigiert einen fehlerhaften internen state automatisch

Das Ziel konnte nicht mit absoluter Sicherheit überprüft werden. Dieses Ziel wurde vermutlich teilweise erreicht, da zum einen über Szenarien belegt wurde, dass der State überschrieben wird, jedoch könnte bei einer Manipulation der Datenstrukturen über Memory Zugriffe der Host-Maschine zur Laufzeit Fehler auftreten, wenn nicht zufällig der State überschrieben wird. Da in dem Projekt keine Kriterien aufgestellt wurden, dass Carbonaut in einer feindlichen Host-Umgebung laufen soll – könnte argumentiert werden, dass dieses Qualitätsziel erreicht wurde.

FL1: Das System besteht aus einer Binary, die flexibel eingesetzt werden kann

Das Ziel wurde erreicht und durch das Szenario 1 in Abschnitt 5.2.1 verifiziert.

FL2: Das System geht davon aus, dass sich die observierte Infrastruktur zur Laufzeit wandelt (vertikale Skalierbarkeit)

Das Ziel wurde erreicht und durch das Szenario 1 in Abschnitt 5.2.1 verifiziert.

FL3: Das System kann mehrfach eingesetzt werden, um eine IT-Infrastruktur zwischen Deployments aufzuteilen (horizontale Skalierbarkeit)

Das Ziel wurde erreicht. Carbonaut kann mit der gleichen Konfiguration mehrfach ausgeführt werden. Dies führt dazu, dass mehr Anfragen beantwortet werden können. Diese Funktionalität wurde in den Szenarien nicht überprüft.

SE1: Authentizität wird außerhalb des Systems konfiguriert und validiert

Das Ziel wurde erreicht.

SE2: Das Projekt verarbeitet keine personenbezogenen Daten

Das Ziel wurde erreicht. Das Projekt gibt auch nicht die Möglichkeit über Plugins personenbezogene Daten zu verarbeiten, da diese im Provider Interface und Datenschema nicht integriert werden können.

R1: Bei unvollständigen Daten wird ein Log erstellt und dann weiterverfahren

Das Ziel wurde erreicht. Bei den Log-Daten des ersten Szenarios `s1-s1-5-carbonaut-log.txt` finden sich die folgenden Einträge. Diese Logeinträge zeigen, dass erst Daten von Ressourcen abgefragt werden, wenn der Status der Ressource auf `aktiv` gesetzt ist. Wird die Ressource in state `provisioning` wird dies übersprungen.

Listing 6.3: Unvollständige Ressourcen werden nicht gespeichert

```
INFO [...] equinix resources not ready resource state=provisioning
INFO [...] equinix resources not ready resource state=provisioning
INFO [...] create new resource resource name=c6ac3a39-X
```

R2: Fehlerhafte Konfiguration wird nicht verarbeitet.

Das Ziel wurde erreicht.

R3: Fehlerhafte Provider-Plugins führen nicht zum Versagen des restlichen Systems.

Das Ziel wurde nicht erreicht. Bei der aktuellen Implementierung kann es dazu führen, dass bei Fehlern das restliche System mit abstürzt. Dies wurde bewusst so entschieden, nachdem in Abschnitt 4.9.2 andere Designvorschläge erarbeitet wurden. In Abschnitt 6.3.1 wird dazu abermals Bezug genommen.

U1: Nutzer können den internen Zustand von Carbonaut abfragen

Das Ziel wurde erreicht. Diese Funktionalität wurde mit der Carbonaut Server Schnittstelle `/static-data` implementiert. Die Schnittstellen sind in der Tabelle 4.8 beschrieben und wurden in den beiden Szenarien in Kapitel 5.1 validiert.

U2: Das Projekt ist in Englisch entwickelt, was eine breitere Öffentlichkeit versteht

Das Ziel wurde erreicht.

6.2.2 U3: Nicht genutzte Konfigurationseinstellungen wird dem Nutzer sichtbar gemacht

Das Ziel wurde nicht erreicht. Optionen zur Verbesserung werden in Abschnitt 6.3.1 besprochen.

U4: Das System kann als Binary oder Container ausgeführt werden

Das Ziel wurde erreicht. In Szenario 1, in Abschnitt 5.2, wurde Carbonaut als Binary ausgeführt. In Szenario 2, in Abschnitt 5.3, wurde Carbonaut als Container in einem Kubernetes Cluster ausgeführt.

U5: Das Projekt ist auf einer Dokumentationswebsite beschrieben

Das Ziel wurde erreicht. Die Dokumentation ist in dem Repository `leonardpahlke/carbonaut-docs` einsehbar und auf Adresse https://carbonaut.dev/ erreichbar.

C1: Carbonaut verändert andere Systeme nicht (nur `READS`, keine `WRITES`)

Das Ziel wurde erreicht. Die beschriebenen Provider Interfaces, in Abschnitt 4.5.4, bilden nur Schnittstellen, die Daten sammeln.

C2: Provider-Interfaces und Datenschema bilden die Interoperabilitätsebene

Das Ziel wurde erreicht.

C3: Container-Images werden für alle gängigen Plattformen erstellt und verteilt

Das Ziel wurde erreicht. Zur Durchführung des Szenario 2 werden Container Images benötigt, die zu Docker Hub[2] gepusht werden. Der Prozess Container images zu bauen ist über das Bash Skript `/hack/container-build-deploy.bash` implementiert. In diesem Skript findet sich das folgende Command `docker buildx build -f Containerfile -platform linux/amd64,linux/arm64,linux/arm/v7,linux/arm/v6 -t $IMAGE_NAME:$TAG -push .` welches mehrere Plattform-Architekturen spezifiziert.

C4: Das System veröffentlicht Daten in einem generischen Schema

Das Ziel wurde erreicht. Das Datenschema wurde in Abschnitt 4.4.4 beschrieben.

C5: Das System ist abwärtskompatibel sein („backward compatibility“) und stellt sicher, dass ältere Versionen weiterhin unterstützt werden

Das Ziel konnte bisher nicht überprüft werden, da nicht mehrere Versionen von Carbonaut vorliegen. Dieses Ziel muss in weiteren Version berücksichtigt werden.

P1: Werden unvollständige Daten verarbeitet, so wartet Carbonaut nicht auf die restlichen Daten, sondern springt zum nächsten Schritt, um die Verarbeitung nicht zu stoppen

Das Ziel wurde erreicht. Zur Begründung siehe 6.2.1.

[2]Docker Hub: https://hub.docker.com/

P1: Werden unvollständige Daten verarbeitet, so wartet Carbonaut nicht auf die restlichen Daten, sondern springt zum nächsten Schritt, um die Verarbeitung nicht zu stoppen

Das Ziel wurde erreicht. Zur Begründung siehe 6.2.1.

FU1: Carbonaut veröffentlicht keine in Teilen unvollständigen Daten

Das Ziel wurde erreicht. In den Szenarien, aus Kapitel 5.1, wurde mehrfach Daten in unterschiedlichen Konfigurationen abgefragt. Die gesammelten Daten waren immer vollständig. Dies wurde mithilfe der generierten Logdaten verifiziert.

FU2: Carbonaut sammelt immer alle Daten angebundener Systeme

Das Ziel wurde erreicht. In den Szenarien, aus Kapitel 5.1, wurde mehrfach Daten in unterschiedlichen Konfigurationen abgefragt. Die gesammelten Daten waren immer vollständig. Dies wurde mithilfe der generierten Logdaten verifiziert.

FU3: Das System sollte alle notwendigen Funktionen bieten, um den vollständigen Anwendungsfall abzudecken, für den es entwickelt wurde

Das Ziel wurde erreicht. Dieses Qualitätsziel jedoch kann über weitere Iterationen von Carbonaut verbessert werden. Die Implementation weiterer Plugins wäre etwa sinnvoll.

E1: Das System fährt bei nicht Benutzung auf ein Minimum herunter (Idle State)

Das Ziel wurde teilweise erreicht. Das System greift auf keine Schnittstellen zurück, wenn dies nicht konfiguriert wurde. In den Szenarien, aus Kapitel 5.1 überprüft. Das System bietet zudem Konfigurationsparameter, die einstellen, wie lange gewartet werden soll, bevor der interne State gespiegelt wird. Standardmäßig ist dieser bei einem `timeout_seconds: 10`.

E2: Das System versucht so wenige Abhängigkeiten zu anderen Projekten und Bibliotheken aufzubauen. Wenn Abhängigkeiten entstehen, dann zu etablierten Projekten, die eine lange Laufzeit generieren können

Das Ziel wurde erreicht. Die Abhängigkeiten sind in der `go.mod` Datei einsehbar. Diese Projekte sind alle frei verfügbar und werden aktiv weiterentwickelt. Wie in Abschnitt 4.8.6 beschrieben wurde, ist weitere Funktionalität in das Carbonaut Projekt verschoben worden, was hätte auch über weitere Abhängigkeiten abgebildet werden können.

E3: Erstellte Schnittstellen und Schemata werden ab dem Release v1 fortführend unterstützt

Das Ziel konnte bisher nicht überprüft werden, da nicht mehrere Versionen von Carbonaut vorliegen. Dieses Ziel muss in weiteren Version berücksichtigt werden.

E4: Das System beinhaltet nur Funktionalität, dass das Kernziel von Carbonaut dient

Das Ziel wurde erreicht.

6.2.3 Übersicht der Zielerreichung

In der nachfolgenden Tabelle wird eine Übersicht aller Qualitätsziele gegeben, die im vorherigen Abschnitt diskutiert wurden. Zu nicht erreichten Qualitätsziele wird in folgenden Abschnitt 6.3.1 Weiterentwicklungsvorschläge gemacht.

Erreichungsgrad	**Qualitätsziele (Referenz)**
Erreicht	M1, M3, M4, SA1, FL1, FL2, SE1, SE2, R1, R2, U1, U2, U4, U5, C1, C2, C3, C4, P1, FU1, FU2, FU3, E2, E4, FL3
Teilweise erreicht	M2, SA2, E1
Nicht erreicht	R3, U3
Kann nicht zurzeit überprüft werden	C5, E3

Tabelle 6.3: Clustering der Qualitätsziele nach Erreichungsgrad

6.3 Weiterentwicklung von Carbonaut

In diesem Abschnitt wird das Carbonaut-Projekt reflektiert. Im vorherigen Abschnitt 6.2 wurden einige Qualitätsziele beschrieben, die bisher nicht erreicht wurden und an die in einer weiteren Iteration angeknüpft werden kann. Zusätzlich werden weitere Beobachtungen und Verbesserungsvorschläge hinzugefügt, die während der Implementierung gemacht wurden.

Nachdem die Erweiterungen besprochen wurden, wird ein größerer Bogen zu den Anfängen der Arbeit geschlagen. Welche Rolle spielt die Idee von Carbonaut im Feld der nachhaltigen Softwareentwicklung? Zudem werden Ideen besprochen, die aus der Entwicklung von Carbonaut hervorgegangen sind.

6.3.1 Zukünftige Erweiterungen

Von den 33 definierten Qualitätszielen wurden 2 bisher nicht erreicht. Für diese wird nun eine Implementierungsstrategie erarbeitet. Weitere Vorschläge werden anschließend gemacht.

R3: Fehlerhafte Provider-Plugins führen nicht zum Versagen des restlichen Systems

Diese Designentscheidung wurde ausführlich in Abschnitt 4.9.2 diskutiert. Die aufgezeigte Option, WebAssembly (WASM) zu nutzen, hat sich als vielversprechend erwiesen, insbesondere im Hinblick auf die Weiterentwicklung von Carbonaut. Diese Option wurde in einem experimentellen Projekt erarbeitet[3] und in Abschnitt 4.9.2 vorgestellt. Die Implementierung dieses Ansatzes würde dank der Integration von Plugins über WASM-Module sicherstellen, dass fehlerhafte Plugins nicht das gesamte Carbonaut-Deployment zum Absturz bringen. Derzeit kann ein Plugin eine `panic()` auslösen, was den gesamten Prozess beendet. Da jedoch alle Plugins Teil der Carbonaut-Codebasis sind, stellt dies aktuell kein Problem dar.

[3] WASM Plugin Projekt: https://github.com/leonardpahlke/example-wasm-pointer-plugins

U3: Nicht genutzte Konfigurationseinstellungen wird dem Nutzer sichtbar gemacht

Diese Funktionalität kann in Carbonaut zügig implementiert werden, ohne andere Pakete abseits von dem `pkg/config`. Der folgende Code zeigt, wie diese Funktionalität implementiert werden kann.

Listing 6.4: U3: Implementationsansatz

```
type Config struct {
    Field1 string `yaml:"field1"`
}

type ConfigWithExtra struct {
    Config    `yaml:",inline"`
    ExtraFields map[string]interface{} `yaml:",inline"`
}

func main() {
    data, _ := io.ReadFile("config.yaml")

    var cfg ConfigWithExtra
    _ = yaml.Unmarshal(data, &cfg)

    for key, value := range cfg.ExtraFields {
        slog.Warn("configuration value has no effect",
            "key", key, "value"m value)
    }
}
```

Plugin Equinix Paging

Das Equinix Plugin gibt paging meta informationen, was in Carbonaut aktuell nicht implementiert ist. Die Implementation würde beinhalten, dass über alle „pages“ iteriert wird, bis null zurückkommt. Alle gesammelten Daten werden akkumuliert zurückgegeben. Die Paging Informationen sind unter den „Meta“-Informationen bereitgestellt. Die

Ergänzungen werden in der `pkg/plugins/staticresplugins/equinix/equix-plugin.go` Datei vorgenommen. Siehe folgenden Ausschnitt.

Listing 6.5: Carbonaut Plugin Equinix Paging Ausschinitt

```
type ProjectDevicesResponse struct {
    Devices []EquinixDevice `json:"devices"`
    Meta struct {
        First struct {
                Href string `json:"href"`
        } `json:"first"`
        Previous interface{} `json:"previous"`
        Self     struct {
                Href string `json:"href"`
        } `json:"self"`
        Next interface{} `json:"next"`
        Last struct {
                Href string `json:"href"`
        } `json:"last"`
        CurrentPage int `json:"current_page"`
        LastPage    int `json:"last_page"`
        Total       int `json:"total"`
    } `json:"meta"`
}
```

Zwischenspeichern von Energie-Mix Daten

In der aktuellen Implementierung werden Energie-Mix-Daten für jede Ressource abgefragt. Dies ist nicht notwendig, da davon ausgegangen werden kann, dass Ressourcen zwar in mehreren Geo-Regionen verstreut verwaltet werden, aber dennoch geclustert sind. Oft befinden sich die meisten Ressourcen in einem zentralen Rechenzentrum, während weitere zur Erhöhung der Ausfallsicherheit oder zu Verbesserung der Latenz Sicherung in anderen Zonen verteilt sind.

Dieses Feature umfasst die Erweiterung des lokalen State zur Clusterung von Ressourcen nach Regionen. `{regionen: {"frankfurt": [r-1, r2], "london": [r-3]}}`. Alle nötigen Informationen liegen bereits vor, der Zugriff auf Endpunkte, muss

angepasst werden. In einem weiteren Schritt wird das Zusammenführen von dynamischen Environment Daten (`dynenv`) von dem Zusammenführen der dynamischen Ressourcendaten (`dynres`) getrennt. Die dynamischen Environment Daten können zudem parallel abgefragt werden.

Diese Änderungen werden in den Dateien `pkg/connector/dynamiccollect.go` und `pkg/connector/state/state.go` vorgenommen.

Erkennen von Aktualisierungen statischer Ressourcen der verbleibenden Ressourcen

In der aktuellen Implementierung werden statische Ressourcendaten (`staticres`) nach der initialen Abfrage nicht wieder erneuert. Dieses könnte jedoch zu Problemen führen bei Ressourcen Informationen, die sich zur Laufzeit ändern können wie die IP-Adresse.

Dies könnte behoben werden, indem statische Daten in dem bestehenden Ablauf in regelmäßigen Abständen den State erneuern. Wie dieser Prozess implementiert ist, wurde in der Laufzeitsicht in Abschnitt 4.6.1 besprochen. Die Änderungen werden in der Datei `pkg/connector/connector.go` vorgenommen.

Implementierung des Prometheus-Exporters

Das Carbonaut-Server-Subsystem ermöglicht, unabhängig von den anderen Subsystemen gesammelte Daten in unterschiedlichen Formaten zu veröffentlichen. Ein weiteres Schema, das sich im Cloud-Native-Kontext anbieten würde, wäre die Implementierung eines Prometheus-Exporters. In Prometheus[4] werden die folgenden Formate unterstützt: *Counter*, *Gauge*, *Histogram*, *Summary*. Die Änderungen werden in der Datei `pkg/server/server.go` vorgenommen.

Weiterentwicklung des Datenschemata

Das in Abschnitt 4.4.4 beschriebene Carbonaut-Datenschema könnte weiter ausgearbeitet werden. Das aktuelle Datenschema folgt einem „Proof-of-Concept“ und minimalistischen Ansatz und wurde daher nicht noch detaillierter ausgearbeitet. Es könnten weitere und

[4] Prometheus: https://prometheus.io/

granularer Daten über laufende Prozesse und Ressourcennutzung gesammelt werden. Eine Fokussierung auf das Netzwerk wäre dabei sinnvoll. Auch Daten über eingesetzte natürliche Ressourcen müssten einbezogen werden, um einen umfassenderen Überblick über die Nachhaltigkeit des Systems zu bieten.

6.3.2 Erkenntnisse und Maßnahmen

In Laufe der Entwicklung des Projektes wurde erkannt, dass zum einen der Entwicklungsansatz zielführend war, aber zugleich geht hervor, dass an zahlreichen Stellen das Projekt weiter verbessert werden kann. Das Carbonaut Projekt gibt einen Rahmen, der weiter ausgearbeitet werden kann. Weitere Plugin Implementierungen, die Entkopplung der Plugins innerhalb des Provider-Plugin Subsystems und die Ausarbeitung und Erweiterung des Datenschemas sind die Hauptaspekte. Der Ansatz und die Idee, die Carbonaut erarbeitet hat, können dabei unverändert bleiben.

Der Anhang enthält einen Auszug aus Szenario 2, der zeigt, welche Daten derzeit erhoben werden. Bei der Betrachtung der Daten wird deutlich, dass diese noch nicht ausreichend sind und weiter aufbereitet werden müssen, um Rückschlüsse zu ziehen. Da die Anwendungsfälle jedoch noch nicht standardisiert sind, müsste zunächst eine Analyse der Rechtslage durchgeführt werden, um festzustellen, welche Informationen z.B. für Jahresabschlüsse erhoben werden müssen. Carbonaut bietet daher eine Vielzahl von Ansätzen für die weitere Forschung.

7 Ausblick und Schluss

In dieser Arbeit wurde untersucht, wie Nachhaltigkeitsmetriken in einem Cloudsystem gesammelt und verarbeitet werden können. Dazu wurde das Projekt Carbonaut entwickelt. Als Basis und Ausgangspunkt wurde zunächst das Thema Nachhaltigkeit im Allgemeinen betrachtet. Nachhaltigkeit wurde als interdisziplinäres Forschungsfeld erkannt, das Einfluss auf die gesamte Gesellschaft und Wirtschaft hat und den Gedanken verfolgt, ökologische, soziale und ökonomische Belange einzubeziehen. Der Begriff Nachhaltigkeit kann auch als Harmonie-Gedanke und -Ansatz verstanden werden.

Anschließend wurde untersucht, welche wechselseitigen Effekte Nachhaltigkeit und Digitalisierung haben, die beide als Transformationsmotoren verstanden werden können. Beide Transformationen beeinflussen sich gegenseitig. Wie diskutiert, ermöglicht die Digitalisierung und Software die Entwicklung neuer Techniken und Verfahren zur Entwicklung nachhaltiger Produkte und Prozesse. Software spielt hier eine zentrale Rolle, um Nachhaltigkeit im größeren Sinne zu gestalten. Dieses Thema wird in der Informatik unter Begriffen wie Green Software untersucht. Es wurde erkannt, dass auch Software auf Nachhaltigkeitskriterien ausgerichtet sein muss.

Ein zentrales Problem von Software ist der Unsichtbarkeitsfaktor, der nicht nur bei ethischen Diskussionen zur Sprache kommt, sondern auch als ein Grund angesehen werden kann, warum es schwierig ist, nachvollziehen zu können, welche Ressourcen gebunden werden, wenn Software entwickelt, vertrieben und eingesetzt wird. Nachhaltigkeit kann in der Softwareentwicklung über Anforderungen definiert werden, es können agile Prozesse angepasst, Designentscheidungen bewusst getroffen oder Architekturmuster verwendet werden, die einem Nachhaltigkeitsgedanken entspringen. Es gibt zahlreiche Forschungsarbeiten, die unterschiedliche Ansätze diskutieren, die in dieser Arbeit aufgegriffen wurden.

Da Server, Rechenzentren, das Internet und Netzwerke einen großen Anteil der globalen Energie verbrauchen und dieser Trend positiv ist, wurde erkannt, dass besonders im

Cloud-Bereich Energie und aufgewendete Ressourcen besser erkannt und in Zusammenhang gebracht werden müssen. Diese Informationen werden durch Abstraktionsschichten nicht natürlich zugänglich gemacht. Es müssen demnach Schnittstellen erweitert und Applikationen eingeführt werden, diese Vermittlung der Informationen zu leisten. Es wurde beschrieben, welche Verantwortung die einzelnen Abstraktionsschichten – Hardware, Betriebssysteme, Benutzeranwendungen, Cluster und Cloud – haben, um effektiv Daten und Aussagen über die Nachhaltigkeit eines Systems treffen zu können.

Diese gesamte Recherche mündete in die Entwicklung des Carbonaut-Projekts. Die Entwicklung von Carbonaut zeigt, dass es nicht trivial ist, automatisiert, effizient und effektiv Daten von einer heterogenen Infrastruktur zu sammeln, um Aussagen über die Nachhaltigkeit eines Systems treffen zu können. Das System wurde in einer detaillierten Architekturdokumentation geplant und anhand dieser implementiert. Die Projektressourcen[1] sind öffentlich zugänglich und auf einer Webseite dokumentiert[2]. In Testszenarien wurde die Qualität und das Erreichen definierter Qualitätsziele verifiziert und diskutiert. Es wurde schließlich erarbeitet, dass das vorgestellte und ausgearbeitete Design dazu dienen kann, Nachhaltigkeitsherausforderungen in der Cloud zu bewältigen. Dabei ist das Projekt ein Baustein eines größeren Systems und wurde nicht entwickelt, um alle Nachhaltigkeitsfragen und -probleme zu lösen. Carbonaut bereitet Informationen auf und legt damit die Datengrundlage, um überhaupt Aussagen und Entscheidungen über die Weiterentwicklung zu treffen.

Zukünftige Arbeiten könnten an den aufgezeigten Weiterentwicklungen von Carbonaut ansetzen. Auch das eingesetzte Datenschema müsste, wie diskutiert, erweitert werden, um ein umfassenderes Bild über die Nachhaltigkeit des Systems zu ermöglichen. Eingesetzte natürliche Ressourcen spielen im Carbonaut-Projekt bislang noch keine Rolle, wären aber eine wünschenswerte Erweiterung. Weitere Arbeiten könnten basierend auf den von Carbonaut gesammelten Daten Analysen durchführen oder Systeme zur Laufzeit anpassen. Alles in allem gibt es vielfältige Möglichkeiten, weitere Forschungsarbeiten zu starten.

[1] Projekt Repository: https://github.com/leonardpahlke/carbonaut/
[2] Projekt Dokumentation: https://carbonaut.dev/

Literaturverzeichnis

[1] AL, Tim L. et: *Global Tipping Points Report 2023*. 2023. – URL https://global-tipping-points.org

[2] AXELROD, Robert ; HAMILTON, William D.: The Evolution of Cooperation. In: *Science* 211 (1981), Nr. 4489, S. 1390–1396. – URL https://www.science.org/doi/abs/10.1126/science.7466396

[3] BANK, World: *The Changing Wealth of Nations 2021: Managing Assets for the Future*. Washington, DC : World Bank, 2021. – URL http://hdl.handle.net/10986/36400. – License: CC BY 3.0 IGO

[4] BATES, Oliver ; LORD, Carolynne ; ALTER, Hayley ; KIRMAN, Ben: Let's start talking the walk: Capturing and reflecting on our limits when working with gig economy workers. In: *Proceedings of the 7th International Conference on ICT for Sustainability*, 2020, S. 227–235

[5] BECKER, Christoph ; BETZ, Stefanie ; CHITCHYAN, Ruzanna ; DUBOC, Leticia ; EASTERBROOK, Steve M. ; PENZENSTADLER, Birgit ; SEYFF, Norbet ; VENTERS, Colin C.: Requirements: The key to sustainability. In: *IEEE Software* 33 (2015), Nr. 1, S. 56–65

[6] BECKER, Christoph ; CHITCHYAN, Ruzanna ; DUBOC, Leticia ; EASTERBROOK, Steve ; PENZENSTADLER, Birgit ; SEYFF, Norbert ; VENTERS, Colin C.: Sustainability design and software: The karlskrona manifesto. In: *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering* Bd. 2 IEEE (Veranst.), 2015, S. 467–476

[7] BEDA, Joe: *Cloud Native Part 1: Definition*. Blog Post. 2016. – URL https://blog.heptio.com/cloud-native-part-1-definition-716ed30e9193

[8] Berkhout, Frans ; Hertin, Julia: Impacts of information and communication technologies on environmental sustainability: Speculations and evidence. In: *Report to the OECD, Brighton* 21 (2001)

[9] Blömeke, Steffen ; Mennenga, Mark ; Herrmann, Christoph ; Kintscher, Lars ; Bikker, Gert ; Lawrenz, Sebastian ; Sharma, Priyanka ; Rausch, Andreas ; Nipprashchk, Mathias ; Goldmann, Daniel u. a.: Recycling 4.0: An integrated approach towards an advanced circular economy. In: *Proceedings of the 7th International Conference on ICT for Sustainability*, 2020, S. 66–76

[10] Book, SOA S.: *What is SOA*. – URL https://collaboration.opengroup.org/projects/soa-book/pages.php?action=show&ggid=1314. – Zugriffsdatum: 15.02.2024

[11] Brockmann, Dirk: *Im Wald vor lauter Bäumen: Unsere komplexe Welt besser verstehen*. Deutscher Taschenbuch Verlag, 2021

[12] Carson, Rachel: *Silent Spring*. Houghton Mifflin, 1962

[13] Castells, Manuel: *The New Economy: Informationalism, Globalization, Networking*. Kap. 2, S. 77–162. In: *The Rise of the Network Society*, John Wiley & Sons, Ltd, 2009. – URL https://onlinelibrary.wiley.com/doi/abs/10.1002/9781444319514.ch2. – ISBN 9781444319514

[14] Chitchyan, Ruzanna ; Becker, Christoph ; Betz, Stefanie ; Duboc, Leticia ; Penzenstadler, Birgit ; Seyff, Norbert ; Venters, Colin C.: Sustainability design in requirements engineering: state of practice. In: *Proceedings of the 38th International Conference on Software Engineering Companion*, 2016, S. 533–542

[15] Christensen, Henrik B.: *The Green Architecture Framework*. 2023. – URL https://baerbak.cs.au.dk/papers/gaf.html. – Zugriffsdatum: 15.02.2024

[16] Commission, European: *The Paris Agreement*. 2019. – URL https://commission.europa.eu/stratefy-and-policy/priorities-2019-2024/european-green-deal_en

[17] Crutchfield, J P. ; Mitchell, M: The evolution of emergent computation. In: *Proceedings of the National Academy of Sciences* 92 (1995), Nr. 23, S. 10742–10746. – URL https://www.pnas.org/doi/abs/10.1073/pnas.92.23.10742

[18] Crutzen, Paul: Geology of mankind. In: *Nature 415, 23* (2002), Jan

[19] Crutzen, Paul ; Stoermer, et a.: *Have we entered the Änthropocene"?* 2010. – URL http://www.igbp.net/news/opinion/opinion/haveweenteredtheanthropocene.5.d8b4c3c12bf3be638a8000578.html

[20] Cunningham, William P. ; Cunningham, Mary A.: *Principles of Environmental Science*. McGraw Hill, 2008

[21] Dick, Markus ; Drangmeister, Jakob ; Kern, Eva ; Naumann, Stefan: Green software engineering with agile methods. In: *2013 2nd international workshop on green and sustainable software (GREENS)* IEEE (Veranst.), 2013, S. 78–85

[22] Donella H. Meadows, et.al: *The Limits to Growth*. Potomac Associates, 1972

[23] Du, Qiang ; Li, Yi ; Bai, Libiao: The Energy Rebound Effect for the Construction Industry: Empirical Evidence from China. In: *Sustainability* 9 (2017), Nr. 5. – URL https://www.mdpi.com/2071-1050/9/5/803. – ISSN 2071-1050

[24] Floridi, Luciano: The fight for digital sovereignty: What it is, and why it matters, especially for the EU. In: *Philosophy & technology* 33 (2020), S. 369–378

[25] Forschung (BMBF), Bundesministerium für Bildung und: *Forschung zur Kreislaufwirtschaft*. https://www.bmbf.de/bmbf/de/forschung/umwelt-und-klima/ressourcen/forschung-zur-kreislaufwirtschaft.html. 2022. – Zugriff am: 02.01.2024

[26] Foundation, Cloud Native C.: *Cloud Native Landscape*. 2024. – URL https://landscape.cncf.io/

[27] Frey, Carl B. ; Osborne, Michael A.: The future of employment: How susceptible are jobs to computerisation? In: *Technological Forecasting and Social Change* 114 (2017), S. 254–280. – URL https://www.sciencedirect.com/science/article/pii/S0040162516302244. – ISSN 0040-1625

[28] G. Lami, M. F.: Software sustainability from a process-centric perspective. In: *European Conference on Software Process Improvement* Springer (Veranst.), 2012, S. 97–108

[29] Gell-Mann, Murray: What is complexity? Remarks on simplicity and complexity by the Nobel Prize-winning author of The Quark and the Jaguar. In: *Complexity* 1 (1995), Nr. 1, S. 16–19. – URL https://onlinelibrary.wiley.com/doi/abs/10.1002/cplx.6130010105

[30] Green, Jessica F.: Private standards in the climate regime: the greenhouse gas protocol. In: *Business and Politics* 12 (2010), Nr. 3, S. 1–37

[31] Grimm, Petra ; Keber, Tobias ; Zöllner, Oliver: *Digitale Ethik. Leben in vernetzten Welten: Reclam Kompaktwissen XL*. Reclam Verlag, 2019

[32] Herrmann, Ulrike: *Das Ende des Kapitalismus: Warum Wachstum und Klimaschutz nicht vereinbar sind–und wie wir in Zukunft leben werden*. Kiepenheuer & Witsch, 2022

[33] Hilker, Thomas ; Lyapustin, Alexei I. ; Tucker, Compton J. ; Hall, Forrest G. ; Myneni, Ranga B. ; Wang, Yujie ; Bi, Jian ; Moura, Yhasmin M. de ; Sellers, Piers J.: Vegetation dynamics and rainfall sensitivity of the Amazon. In: *Proceedings of the National Academy of Sciences* 111 (2014), Nr. 45, S. 16041–16046. – URL https://www.pnas.org/doi/abs/10.1073/pnas.1404870111

[34] Hindle, Abram: Green software engineering: the curse of methodology. In: *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)* Bd. 5 IEEE (Veranst.), 2016, S. 46–55

[35] Initiative, Life C.: *What is Life Cycle Thinking?* https://www.lifecycleinitiative.org/starting-life-cycle-thinking/what-is-life-cycle-thinking/

[36] James, H M.: What is computer ethics. In: *Metaphilosophy* 16 (1985), Nr. 4, S. 266–275

[37] Khan, Kashif N. ; Hirki, Mikael ; Niemi, Tapio ; Nurminen, Jukka K. ; Ou, Zhonghong: Rapl in action: Experiences in using rapl for power measurements. In: *ACM Transactions on Modeling and Performance Evaluation of Computing Systems (TOMPECS)* 3 (2018), Nr. 2, S. 1–26

[38] Khazzoom, J D.: Economic implications of mandated efficiency in standards for household appliances. In: *The energy journal* 1 (1980), Nr. 4, S. 21–40

[39] Lago, Patricia ; Jansen, Toon: Creating environmental awareness in service oriented software engineering. In: *Service-Oriented Computing: ICSOC 2010 International Workshops, PAASC, WESOA, SEE, and SOC-LOG, San Francisco, CA, USA, December 7-10, 2010, Revised Selected Papers 8* Springer (Veranst.), 2011, S. 181–186

[40] Lovelock, James E. ; Margulis, Lynn: Atmospheric homeostasis by and for the biosphere: the gaia hypothesis. In: *Tellus A: Dynamic Meteorology and Oceanography* (1974), Jan

[41] Marx, Karl: *Das Kapital*. Hamburg, Deutschland : Verlag von Otto Meissner, 1867

[42] Masanet, Eric ; Authors, Other: Recalibrating global data center energy-use estimates. In: *Science* 367 (2020), Nr. 6481. – URL https://datacenters.lbl.gov/sites/default/files/Masanet_et_al_Science_2020.full_.pdf

[43] Max Craglia, et.al.: Digital Earth 2020: towards the vision for the next decade. In: *International Journal of Digital Earth* 5 (2012), Nr. 1, S. 4–21. – URL https://doi.org/10.1080/17538947.2011.638500

[44] Maycroft, Neil e.: Consumption, planned obsolescence and waste. (2009)

[45] Münzel, Harald: Towards an ethical foundation of green software engineering. In: *2015 IEEE 10th International Conference on Global Software Engineering Workshops* IEEE (Veranst.), 2015, S. 23–26

[46] Nations), Vereinten Nationen (.: *Millenniumsentwicklungsziele (MDGs)*. 2000. – URL https://www.bmz.de/de/service/lexikon/mdg-millenniumsentwicklungsziele-mdgs-14674

[47] Nations), Vereinten Nationen (.: *United Nations Conference on Sustainable Development, Rio+20*. 2012. – URL https://sustainabledevelopment.un.org/rio20.html

[48] Nations), Vereinten Nationen (.: *The Paris Agreement*. 2015. – URL https://www.un.org/en/climatechange/paris-agreement

[49] Nations), Vereinten Nationen (.: *The global E-Waste Monitor 2020*. Global E-Waste Monitor. 2020. – URL https://ewastemonitor.info/wp-content/uploads/2020/11/GEM_2020_def_july1_low.pdf

[50] Naumann, Stefan ; Dick, Markus ; Kern, Eva ; Johann, Timo: The GREENSOFT Model: A reference model for green and sustainable software and its engineering. In: *Sustainable Computing: Informatics and Systems* 1 (2011), Nr. 4, S. 294–304

[51] Naumann, Stefan ; Kern, Eva ; Dick, Markus ; Johann, Timo: Sustainable software engineering: Process and quality models, life cycle, and social aspects. In: *ICT Innovations for Sustainability* Springer (Veranst.), 2015, S. 191–205

[52] Norris, Pippa: *Digital divide: Civic engagement, information poverty, and the Internet worldwide*. Cambridge university press, 2001

[53] Ostrom, Elinor: *Governing the commons: The evolution of institutions for collective action*. Cambridge university press, 1990

[54] Parker, Geoffrey G. ; Van Alstyne, Marshall W. ; Choudary, Sangeet P.: *Platform revolution: How networked markets are transforming the economy and how to make them work for you*. WW Norton & Company, 2016

[55] Piketty, Thomas: *Capital in the Twenty-First Century*. Cambridge, MA, USA : Harvard University Press, 2014

[56] Polanyi, Karl: *The Great Transformation*. New York, USA : Farrar & Rinehart, 1944

[57] Pouri, Maria J. ; Hilty, Lorenz M.: The relevance of digital sharing business models for sustainability. In: *Proceedings of the 7th International Conference on ICT for Sustainability*, 2020, S. 77–87

[58] Precht, Richard D.: *Jäger, Hirten, Kritiker: Eine Utopie für die digitale Gesellschaft*. Goldmann Verlag, 2018

[59] Rosa, Hartmut: *Resonanz: Eine soziologie der weltbeziehung*. Suhrkamp verlag, 2016

[60] Smil, Vaclav: *Wie die Welt wirklich funktioniert: Die fossilen Grundlagen unserer Zivilisation und die Zukunft der Menschheit*. CH Beck, 2023

[61] Steffen, Will ; Richardson, Katherine ; Rockström, Johan ; Cornell, Sarah E. ; Fetzer, Ingo ; Bennett, Elena M. ; Biggs, Reinette ; Carpenter, Stephen R. ; De Vries, Wim ; De Wit, Cynthia A. u. a.: Planetary boundaries: Guiding human development on a changing planet. In: *Science* 347 (2015), Nr. 6223, S. 1259855

[62] Svetinovic, Davor: Strategic requirements engineering for complex sustainable systems. In: *Systems Engineering* 16 (2013), Nr. 2, S. 165–174

[63] Tarbuck, Edward J. ; Lutgens, Frederick K. ; Tasa, Dennis G.: *Earth Science*. 15. Pearson, 2018. – ISBN 9780134543536

[64] Trittin-Ulbrich, Hannah ; Scherer, Andreas G. ; Munro, Iain ; Whelan, Glen: Exploring the dark and unexpected sides of digitalization: Toward a critical agenda. In: *Organization* 28 (2021), Nr. 1, S. 8–25. – URL https://doi.org/10.1177/1350508420968184

[65] Twenge, Jean M. ; Campbell, W K.: *The narcissism epidemic: Living in the age of entitlement*. Simon and Schuster, 2009

[66] Umweltbundesamt: *Fallstudie: Seltene Erden in China - Bayan Obo*. 2015. – URL https://www.umweltbundesamt.de/sites/default/files/medien/378/dokumente/umsoress_fallstudie_seltene_erden_china_bayan_obo.pdf

[67] UN Secretary General, World Commission on E. ; Development: *Our Common Future, Report of the World Commission on Environment and Development (Brundtland report)*. 1987. – URL https://digitallibrary.un.org/record/139811

[68] Von Carlowitz, Hans C.: *Sylvicultura oeconomica*. Braun, 1732

[69] Wirth, Niklaus: A plea for lean software. In: *Computer* 28 (1995), Nr. 2, S. 64–68

[70] World Resources Institute and World Business Council for Sustainable Development: *Greenhouse Gas Protocol*. 2023. – URL https://ghgprotocol.org/. – [Online; accessed 29-Jan-2024]

[71] Wynne, Astrid ; Kenny, Rich: Limitations of linear GHG Protocol carbon reporting in achieving circular progress. In: *E3S Web of Conferences* Bd. 455 EDP Sciences (Veranst.), 2023, S. 01013

[72] Zeyu Yang, Wesley A.: Part-time Power Measurements: nvidia-smi's Lack of Attention. (2024). – URL https://arxiv.org/html/2312.02741v2

A Anhang

A.1 Carbonaut Test Auszüge

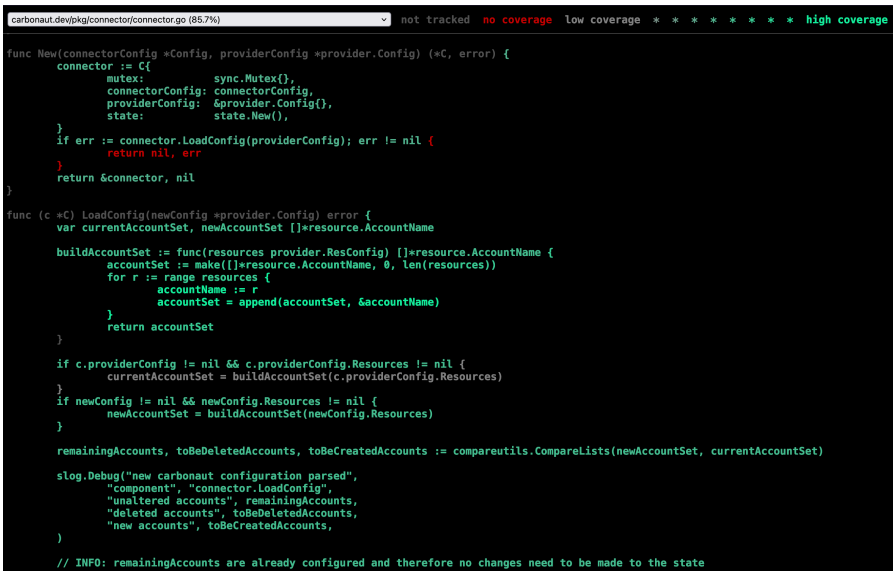

Abbildung A.1: Beispiel für die Visualisierung der Testabdeckung

Listing A.1: Carbonaut Testing Ausgabe

```
?       carbonaut.dev [no test files]
?       carbonaut.dev/hack/configfile-example   [no test files]
?       carbonaut.dev/pkg/config [no test files]
goos: darwin
goarch: arm64
pkg: carbonaut.dev/pkg/connector
BenchmarkCollect-8                 225543              5298 ns/op
BenchmarkLoadConfig
BenchmarkLoadConfig-8              267580              4579 ns/op
```

```
PASS
ok      carbonaut.dev/pkg/connector     3.629s
?       carbonaut.dev/pkg/connector/state       [no test files]
?       carbonaut.dev/pkg/plugin/dynenvplugins [no test files]
PASS
ok      carbonaut.dev/pkg/plugin/dynenvplugins/electricitymaps  0.138s
?       carbonaut.dev/pkg/plugin/dynenvplugins/mockenergymix
        [no test files]
?       carbonaut.dev/pkg/plugin/dynresplugins  [no test files]
?       carbonaut.dev/pkg/plugin/dynresplugins/mockenergy
        [no test files]
PASS
ok      carbonaut.dev/pkg/plugin/dynresplugins/scaphandre       0.110s
?       carbonaut.dev/pkg/plugin/staticresplugins [no test files]
PASS
ok      carbonaut.dev/pkg/plugin/staticresplugins/equinixplugin 0.100s
?       carbonaut.dev/pkg/plugin/staticresplugins/mockcloudplugin
        [no test files]
?       carbonaut.dev/pkg/provider [no test files]
?       carbonaut.dev/pkg/provider/environment  [no test files]
?       carbonaut.dev/pkg/provider/plugin       [no test files]
?       carbonaut.dev/pkg/provider/resource     [no test files]
?       carbonaut.dev/pkg/provider/topology     [no test files]
?       carbonaut.dev/pkg/provider/types/dynenv [no test files]
?       carbonaut.dev/pkg/provider/types/dynres [no test files]
?       carbonaut.dev/pkg/provider/types/staticres      [no test files]
?       carbonaut.dev/pkg/server [no test files]
goos: darwin
goarch: arm64
pkg: carbonaut.dev/pkg/util/cache
BenchmarkCacheGetExpiring
BenchmarkCacheGetExpiring-8                     26694877
43.94 ns/op
BenchmarkCacheGetNotExpiring
BenchmarkCacheGetNotExpiring-8                  100000000
11.40 ns/op
```

```
BenchmarkRWMutexMapGet
BenchmarkRWMutexMapGet-8                          93270588
10.83 ns/op
BenchmarkRWMutexInterfaceMapGetStruct
BenchmarkRWMutexInterfaceMapGetStruct-8           68154868
17.68 ns/op
BenchmarkRWMutexInterfaceMapGetString
BenchmarkRWMutexInterfaceMapGetString-8           86841148
13.79 ns/op
BenchmarkCacheGetConcurrentExpiring
BenchmarkCacheGetConcurrentExpiring-8             13811670
87.48 ns/op
BenchmarkCacheGetConcurrentNotExpiring
BenchmarkCacheGetConcurrentNotExpiring-8          16151419
73.10 ns/op
BenchmarkRWMutexMapGetConcurrent
BenchmarkRWMutexMapGetConcurrent-8                16792699
73.42 ns/op
BenchmarkCacheGetManyConcurrentExpiring
BenchmarkCacheGetManyConcurrentExpiring-8         25984992
86.66 ns/op
BenchmarkCacheGetManyConcurrentNotExpiring
BenchmarkCacheGetManyConcurrentNotExpiring-8      45282948
75.75 ns/op
BenchmarkCacheSetExpiring
BenchmarkCacheSetExpiring-8                       21081309
56.96 ns/op
BenchmarkCacheSetNotExpiring
BenchmarkCacheSetNotExpiring-8                    51067723
23.52 ns/op
BenchmarkRWMutexMapSet
BenchmarkRWMutexMapSet-8                          61847565
19.86 ns/op
BenchmarkCacheSetDelete
BenchmarkCacheSetDelete-8                         23646717
50.45 ns/op
```

```
BenchmarkRWMutexMapSetDelete
BenchmarkRWMutexMapSetDelete-8                  27455634
43.94 ns/op
BenchmarkCacheSetDeleteSingleLock
BenchmarkCacheSetDeleteSingleLock-8             43985313
27.60 ns/op
BenchmarkRWMutexMapSetDeleteSingleLock
BenchmarkRWMutexMapSetDeleteSingleLock-8        46137651
26.10 ns/op
BenchmarkIncrementInt
BenchmarkIncrementInt-8                         36793544
32.30 ns/op
BenchmarkDeleteExpiredLoop
BenchmarkDeleteExpiredLoop-8                        1688
700870 ns/op
PASS
ok      carbonaut.dev/pkg/util/cache    27.714s
PASS
ok      carbonaut.dev/pkg/util/compareutils     0.110s
PASS
ok      carbonaut.dev/pkg/util/freeport 0.076s
PASS
ok      carbonaut.dev/pkg/util/httpwrapper      0.097s
goos: darwin
goarch: arm64
pkg: carbonaut.dev/pkg/util/logger
BenchmarkLog
go:29 | benchmarking i=5223
BenchmarkLog-8              158134              6439 ns/op
PASS
ok      carbonaut.dev/pkg/util/logger   7.893s
PASS
ok      carbonaut.dev/pkg/util/promscraper      0.133s
```

A.2 Carbonaut Szenario 1 - State Beispiel

Diese Daten wurden im Rahmen des Szenarios 1 in Schritt `s1-s3-2` erhoben. Das Szenario ist in Abschnitt 5.2.1 beschrieben.

Listing A.2: Szenario 1: s1-s3-2 - interner State mit statischen Daten


```
{
   "accounts":{
      "1":{
         "name":"equinix",
         "projects":{
            "1":{
               "name":"605f7ac9-263f-4fb7-9b1c-223970b4d920",
               "resources":{
                  "1":{
                     "name":"4ce59185-7b6e-4345-903e-2944705be7bc",
                     "static_data":{
                        "id":"4ce59185-7b6e-4345-903e-2944705be7bc",
                        "user":"root",
                        "os":{
                           "version":"12",
                           "distro":"debian",
                           "name":"Debian 12"
                        },
                        "ipv4":"145.40.94.191",
                        "cpus":[
                           {
                              "count":1,
                              "type":"Intel Xeon E-2278G\
                                8-Core Processor @ 3.40GHz",
                              "cores":"8",
                              "threads":"16",
                              "speed":"3.40GHz",
                              "arch":"x86",
                              "model":"E-2278G",
                              "manufacturer":"Intel",
```

```
                        "name":"Intel Xeon E-2278G Processor"
                    }
                ],
                "gpus":[
                    {
                        "count":1,
                        "type":"Intel HD Graphics P630"
                    }
                ],
                "nics":[
                    {
                        "count":2,
                        "type":"10Gbps"
                    }
                ],
                "drives":[
                    {
                        "count":2,
                        "type":"SSD",
                        "size":"480GB"
                    }
                ],
                "memory_gb":"32GB",
                "location":{
                    "city":"Frankfurt",
                    "country":"DE",
                    "address":"Kruppstrasse 121-127",
                    "zip_code":"60388",
                    "code":"fr2"
                }
            },
            "created_at":"2024-06-10T12:07:59.671407+02:00",
            "plugin":"equinix"
        }
    },
    "created_at":"2024-06-10T12:07:59.258514+02:00"
```

```
            }
        },
        "created_at":"2024-06-10T12:07:58.454723+02:00"
    }
  }
}
```


A.3 Carbonaut Szenario 1 - Metriken Beispiel

Diese Daten wurden im Rahmen des Szenarios 1 in Schritt `s1-s3-3` erhoben. Das Szenario ist in Abschnitt 5.2.1 beschrieben.

Listing A.3: Szenario 1: s1-s3-3 - Metriken mit dynamischen und statischen Daten


```
{
  "equinix":{
    "605f7ac9-263f-4fb7-9b1c-223970b4d920":{
      "4ce59185-7b6e-4345-903e-2944705be7bc":{
        "dynamic_data":{
          "res_data":{
            "cpu_frequency":4603,
            "energy_host_mirojoules":0,
            "cpu_load_percentage":0
          },
          "env_data":{
            "solar_percentage":49.50979603007228,
            "wind_percentage":13.550235845259751,
            "hydro_percentage":4.201748313666146,
            "nuclear_percentage":0.17064571695349098,
            "geothermal_percentage":0.033807170339842554,
            "gas_percentage":9.520743113801375,
            "oil_percentage":0.49905822882624723,
            "biomass_percentage":7.8835101502004274,
            "coal_percentage":14.155545180868362,
            "other_sources_percentage":0.474910250012074,
            "fossil_fuels_percentage":24.345992240449473,
```

```
            "renewable_percentage":75.17909750953845
        }
    },
    "static_data":{
        "id":"4ce59185-7b6e-4345-903e-2944705be7bc",
        "user":"root",
        "os":{
            "version":"12",
            "distro":"debian",
            "name":"Debian 12"
        },
        "ipv4":"145.40.94.191",
        "cpus":[
            {
                "count":1,
                "type":"Intel Xeon E-2278G\
                   8-Core Processor @ 3.40GHz",
                "cores":"8",
                "threads":"16",
                "speed":"3.40GHz",
                "arch":"x86",
                "model":"E-2278G",
                "manufacturer":"Intel",
                "name":"Intel Xeon E-2278G Processor"
            }
        ],
        "gpus":[
            {
                "count":1,
                "type":"Intel HD Graphics P630"
            }
        ],
        "nics":[
            {
                "count":2,
                "type":"10Gbps"
```

```
                    }
                ],
                "drives":[
                    {
                        "count":2,
                        "type":"SSD",
                        "size":"480GB"
                    }
                ],
                "memory_gb":"32GB",
                "location":{
                    "city":"Frankfurt",
                    "country":"DE",
                    "address":"Kruppstrasse 121-127",
                    "zip_code":"60388",
                    "code":"fr2"
                }
            }
        }
    }
}
```

Erklärung zur selbstständigen Bearbeitung

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort	Datum	Unterschrift im Original