hochschule mannheim

Verteiltes Monitoring in Cloud-Native-Umgebungen - Analyse und Integration

Sascha Nockel

Bachelor-Thesis

zur Erlangung des akademischen Grades Bachelor of Science (B.Sc.)

Studiengang Informatik

Fakultät für Informatik

Hochschule Mannheim

30.08.2021

Betreuer

Prof. Dr. Thomas Specht, Hochschule Mannheim

Stephan Hochdörfer, bitExpert AG

Nockel, Sascha:
Verteiltes Monitoring in Cloud-Native-Umgebungen - Analyse und Integration / Sascha Nockel. –
Bachelor-Thesis, Mannheim: Hochschule Mannheim, 2021. 96 Seiten.

Nockel, Sascha:
Distributed Monitoring in Cloud-Native Environments - Analysis and Integration / Sascha Nockel. –
Bachelor Thesis, Mannheim: University of Applied Sciences Mannheim, 2021. 96 pages.

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Ich bin damit einverstanden, dass meine Arbeit veröffentlicht wird, d. h. dass die Arbeit elektronisch gespeichert, in andere Formate konvertiert, auf den Servern der Hochschule Mannheim öffentlich zugänglich gemacht und über das Internet verbreitet werden darf.

Mannheim, 30.08.2021

Sascha Nockel

Abstract

Verteiltes Monitoring in Cloud-Native-Umgebungen - Analyse und Integration

Das Ziel dieser Arbeit ist die Konzeption und Integration einer Monitoring Lösung für die private Cloud der bitExpert AG. Anspruch ist, die Lösung nach den aktuellen Standards im Bereich Cloud Computing zu gestalten. Hierfür wird Prometheus in Verbindung mit Thanos verwendet, um Ausfallsicherheit und Skalierbarkeit zu gewährleisten. Um die Anforderungen an die Lösung zu bestimmen, werden Experteninterviews durchgeführt und außerdem Randbedingungen ermittelt sowie die verschiedenen Betriebs- und Integrationsmöglichkeiten evaluiert. Anhand dieser Ergebnisse wird ein Systementwurf erarbeitet und anschließend unter Verwendung aktueller Cloud Technologien umgesetzt. Am Ende der Arbeit steht eine Evaluation der Umsetzung, in der klar wird, welche Anforderungen erfüllt werden und welche nicht.

Distributed Monitoring in Cloud-Native Environments - Analysis and Integration

The goal of this work is the conception and integration of a monitoring solution for the private cloud of bitExpert AG. The aim is to design the solution according to the current standards in the field of cloud computing. For this, Prometheus is used in conjunction with Thanos to ensure reliability and scalability. To determine the requirements for the solution, expert interviews are conducted, constraints are determined and the various deployment and integration options are evaluated. Based on these results, a system design is developed and then implemented using current cloud technologies. At the end of this work there is an evaluation of the implementation in which it becomes clear which requirements are fulfilled and which are not.

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Motivation

Als Dienstleister im Bereich der digitalen Transformation nutzt die bitExpert AG für interne Tools wie auch für Kundenprojekte das Infrastructure as a Service (IaaS) Angebot des Cloud Anbieters IONOS. Es gelang der Umstieg von einem traditionellen Setup mit virtuellen Maschinen hin zu einer Cloud unter Verwendung von Nomad, einer Lösung zur Orchestrierung von Docker Containern. Um die Cloud Server sowie die Docker Instanzen überwachen zu können, soll Prometheus in Verbindung mit Thanos eingesetzt werden, um eine ausfallsichere und skalierbare Monitoring Lösung zu implementieren. Prometheus wurde gewählt, weil es im Cloud Computing Bereich als der inoffizielle Standard gilt, um Services und Systeme zu überwachen. Außerdem bieten viele Produkte im Technologie Stack der bitExpert AG bereits native Schnittstellen für Prometheus, darunter befinden sich unter anderem die Orchestrierungslösung Nomad, die Service Mesh Lösung Consul und der Traefik Reverse Proxy.

1.2 Zielsetzung

Das Ziel dieser Arbeit ist die Konzeption und Integration der Monitoring Lösung Prometheus in die bestehende IT-Infrastruktur der bitExpert AG unter Berücksichtigung der Aspekte High Availability (HA) und Sicherheit. Neben der Überwachung der internen Cloud soll die Überwachung externer Kundensysteme ebenfalls möglich sein. Die Lösung soll es außerdem ermöglichen, die verschiedenen Entwick-

lungsteams bei ihrer täglichen Arbeit zu unterstützen, indem den Teams Zugang zu den für sie relevanten Metriken und Benachrichtigungen eingeräumt werden kann.

1.3 Aufbau der Arbeit

In Kapitel 2 dieser Arbeit werden die Grundlagen für die spätere Konzeption und Umsetzung vermittelt. Unter anderem werden Begrifflichkeiten und Konzepte wie zum Beispiel IaaS oder Continuous Integration (CI) und Continuous Delivery (CD) erklärt. Außerdem werden die bei der Umsetzung verwendeten Technologien vorgestellt und deren Funktionsweise sowie Zusammenspiel erläutert.

Kapitel 3 beschreibt die Vorgehensweise für die Anforderungsanalyse, die Erarbeitung der Systemarchitektur, die Herangehensweise für den Machbarkeitsnachweis sowie die abschließende Evaluation der Umsetzung.

Die Grundlagen für die spätere Konzeption werden in Kapitel 4 anhand der durchgeführten Experteninterviews geschaffen. Hierfür werden aus den Experteninterviews Anwendungsfälle, funktionale Anforderungen sowie Randbedingungen und Qualitätsanforderungen abgeleitet.

Darauf folgt die Analyse des Ist-Zustands der IT-Infrastruktur und der Integrations- und Betriebsmöglichkeiten in Kapitel 5. Anhand der so ermittelten Informationen wird der Systementwurf für die Monitoring Lösung erarbeitet und erläutert. Hier wird außerdem auf Details wie das Konfigurationsmanagement eingegangen.

Die Umsetzung des Systementwurfs wird schließlich in Kapitel 6 dargestellt. So wird zum Beispiel erklärt, wie die verschiedenen Komponenten mithilfe von CD/CD Pipelines auf die Systeme verteilt und deren Konfigurationen zentral gesteuert werden. Zusätzlich wird das Zusammenspiel der Komponenten sowie die Verwendung der Monitoring Lösung erklärt. Hierfür werden die grundlegenden Funktionalitäten wie etwa der Ablauf beim Absetzen einer Prometheus Query Language (PromQL) Abfrage, oder der Kommunikationsfluss bei einem Alert dargestellt und erläutert.

In Kapitel 7 wird evaluiert, inwiefern die Umsetzung des Systementwurfs die ermittelten Anforderungen aus Kapitel 4 erfüllt. Es wird außerdem darauf eingegangen wie die Anforderungen erfüllt werden, beziehungsweise erklärt, wieso sie nicht vollständig erfüllt werden können.

Abschließend werden die Ergebnisse der Arbeit in Kapitel 8 zusammengefasst und ein Ausblick für die zukünftige Entwicklung der Monitoring Lösung gegeben, beziehungsweise aufgezeigt, wo noch Potenzial für Verbesserungen besteht.

Kapitel 2

Grundlagen

In diesem Kapitel werden die Grundlagen für die spätere Analyse der Infrastruktur sowie die Konzeption der Monitoring Lösung erläutert. Es werden unter anderem Begrifflichkeiten wie IaaS und HA beschrieben sowie die für die Monitoring Lösung verwendeten Komponenten vorgestellt.

Außerdem soll dieses Kapitel dazu dienen, den aktuellen Technologie Stack der bitExpert AG vorzustellen, da die aktuelle Infrastruktur mithilfe des IaaS Angebots des Public Cloud Anbieters IONOS[1] in Kombination mit verschiedenen Technologien wie Nomad oder Consul betrieben wird.

2.1 High Availability und Cloud Computing

2.1.1 High Availability

Der Betrieb von Diensten in einem HA Setup stellt eine besondere Herausforderung dar. Während die meisten Hardwareausfälle durch das Ersetzen oder Vorhalten von Ersatzkomponenten, welche bei Ausfällen übernehmen können, in den Griff zu bekommen sind, muss Software genau für diesen Anwendungsfall optimiert werden. Außerdem ist Software in der Regel abhängig von der Umgebung, in der sie betrieben wird. So bedeutet der Ausfall des physikalischen Netzwerks oder der Festplatte normalerweise ebenfalls den Ausfall des Dienstes, da dieser dann nicht mehr erreichbar ist. Besonders mit dem Ausfall von Hardware muss deshalb immer gerechnet werden, denn Untersuchungen zeigen, dass besonders Ausfälle von Festplatten

[1] https://cloud.ionos.de/

weniger die Ausnahme als viel mehr die Regel sind (Bairavasundaram u. a. 2008). Genau deswegen muss beim Entwurf eines HA Setups, beziehungsweise einer Anwendung die hochverfügbar betrieben werden soll, jede Komponente vom Rechner über das Netzwerk bis hin zur Internetverbindung berücksichtigt werden.

Eines der bekanntesten Design Prinzipien zum Steigern der Verfügbarkeit ist das Eliminieren des Single Point of Failure (SPOF). Ein SPOF kann sich auf Software- oder auch Hardwareebene befinden, so wäre ein typisches Beispiel auf Hardwareebene der Ausfall einer Netzwerkkarte oder der Festplatte. Hier lässt sich Ausfallsicherheit durch eine redundante Netzwerkkarte oder das Verwenden von Redundant Array of Inexpensive Disks (RAID) erreichen (Ahluwalia und Jain 2006).

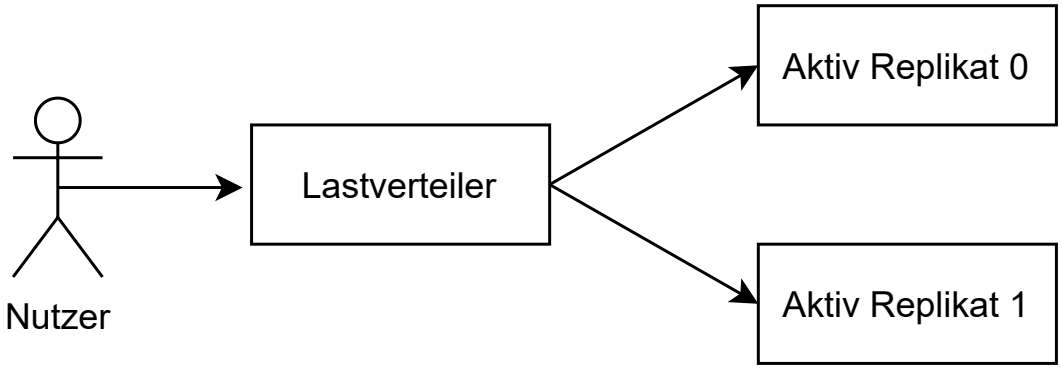

Abbildung 2.1: Schematische Darstellung der Aktiv-Aktiv-Redundanz (Ahluwalia und Jain 2006).

Auf der Softwareebene sind die Lösungsansätze meist komplexer umzusetzen. Hier kann beispielsweise mit mehreren Replikationen des Dienstes auf verschiedenen Rechnerknoten gearbeitet werden. Diese Replikate können dann alle zusammen die Aufgaben des Dienstes wahrnehmen (Aktiv-Aktiv-Redundanz) und Aufgaben über einen Lastverteiler zugeteilt bekommen wie in Abbildung 2.1 dargestellt. Eine solche Architektur bietet sich vor allem an, wenn es sich um zustandslose Dienste wie etwa statische Webseiten oder Webservices handelt. Trotzdem sollte der Lastverteiler benachrichtigt werden, sobald ein Replikat nicht mehr erreichbar ist, damit weitere Anfragen nicht mehr an das ausgefallene Replikat gesendet werden. Moderne Lastverteiler wie zum Beispiel der Traefik Proxy (Abschnitt 2.5) können diesen sogenannten “Health Check” selbst durchführen beziehungsweise Anfragen bei ausbleibender Antwort automatisch an ein anderes Replikat weiterleiten (Ahluwalia und Jain 2006).

Sollte es sich um eine zustandsbehaftete Anwendung wie zum Beispiel eine Datenbank handeln, ist es besonders in klassischen Architekturen mit festen Rechnerkno-

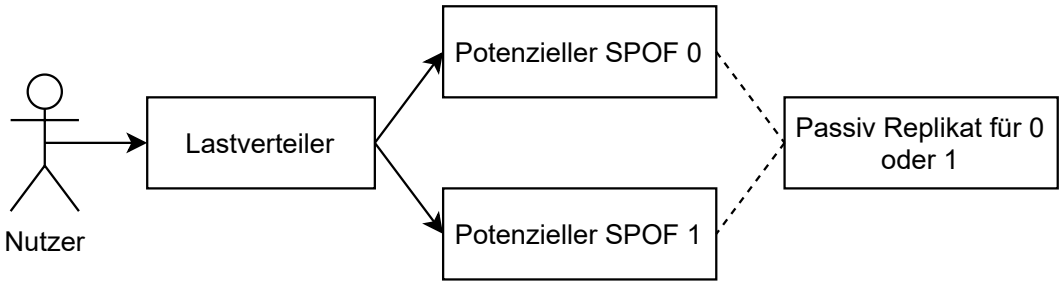

Abbildung 2.2: Schematische Darstellung der Aktiv-Passiv-Redundanz (Ahluwalia und Jain 2006).

ten oft nicht möglich, Anfragen beim Ausfall eines Knotens an ein beliebiges Replikat weiterzuleiten. In einer solchen Situation ist es nötig, ein oder mehrere passive Replikate vorzuhalten wie in Abbildung 2.2 dargestellt (Aktiv-Passiv-Redundanz). So kann beim Ausfall einer aktiven Instanz die passive Instanz übernehmen, auch wenn dies Ressourcen bindet, welche die meiste Zeit nicht verwendet werden (Drake u. a. 2005; Ahluwalia und Jain 2006).

Der Begriff HA wird im IT-Umfeld in der Regel im Zusammenhang mit Diensten und Systemen verwendet, welche in besonders hohem Maß zur Verfügung stehen beziehungsweise besonders ausfallsicher sind oder sein sollen. Das kann eine gemietete Virtuelle Maschine (VM), gemieteter Speicherplatz in der Cloud oder auch ein im eigenen Rechenzentrum betriebener Dienst sein. Die Verfügbarkeit eines solchen Dienstes oder Systems wird üblicherweise in der Anzahl der Neunen nach den 99% angegeben, der Zeitraum bezieht sich meist auf ein Jahr. So würde eine Verfügbarkeit von 99,9% (3NINES) eine Ausfallzeit des Dienstes von circa neun Stunden für das gesamte Jahr bedeuten, wobei es bei 99,99% (4NINES) eine Stunde und bei 99,999% (5NINES) nur noch fünf Minuten im Jahr wären (Ahluwalia und Jain 2006).

Bei Diensten wie Amazon Web Services (AWS) oder Microsoft Azure legt der Anbieter diese garantierten Verfügbarkeiten und eventuelle Strafen bei Nichteinhaltung in der Regel in seinem Service Level Agreement (SLA) fest. Je besser die Verfügbarkeit sein soll, desto komplexer und in der Regel teurer, wird jedoch meist auch der Betrieb. Da in der heutigen Zeit der Großteil des alltäglichen Lebens und Arbeitens von Cloud-Diensten abhängt, hat HA vor allem für Unternehmen eine große Relevanz. Ein Ausfall von kritischen Diensten kann das Alltagsgeschäft lahmlegen und immense Kosten verursachen oder sogar Menschenleben gefährden, falls kritische Infrastrukturen betroffen sind. Als Beispiel dafür ist die Krankenversorgung anzuführen (Mesbahi, Rahmani und Hosseinzadeh 2018).

2.1.2 Infrastructure as a Service

Besonders für kleine bis mittelständische IT-Unternehmen ist der Schritt in die Cloud oft interessant. Dies liegt vor allem daran, dass für den Betrieb von Diensten keine initialen Kosten für Hardware anfallen, auch die Personalkosten halten sich in Grenzen, da keine eigene physikalische Infrastruktur verwaltet und gepflegt werden muss. Gerade wenn ein Unternehmen noch keine eigene Infrastruktur betreibt, sind diese beiden Faktoren ausschlaggebend, während Unternehmen, die bereits ein eigenes Rechenzentrum besitzen, nur bedingt von der Cloud profitieren (Tak, Urgaonkar und Sivasubramaniam 2011).

Die beiden Hauptargumente für die Infrastruktur in der Cloud sind neben den entfallenden Kosten für Hardware und den geringeren Kosten für Personal, die Abrechnung nach Nutzung und die Elastizität (Serrano, Gallardo und Hernantes 2015). Während ein selbst verwaltetes Rechenzentrum in der Regel so ausgelegt ist, dass der "Worst Case", also die größte zu erwartende Last, verkraftet werden kann, ist dies bei Cloud Angeboten nicht nötig, da Rechenleistung je nach Bedarf hinzugebucht oder wieder freigegeben werden kann. So werden weniger Ressourcen gebunden, die gerade nicht gebraucht werden. Deshalb sind gerade Dienste mit stark variierenden Auslastungen meist gut für die Cloud geeignet, da eine automatische Skalierung theoretisch unbegrenzt möglich ist (Tak, Urgaonkar und Sivasubramaniam 2011). Auch in Deutschland geht der Trend deshalb weg vom eigenen Rechenzentrum und hin in Richtung IaaS (ISG 2019).

Bei IaaS handelt es sich um eine Dienstleistung, bei der ein Anbieter seine eigene Infrastruktur mithilfe von Software über das Internet verfügbar macht (IONOS 2021a). Die komplette Infrastruktur, Rechner, Router, Firewalls etc. wird virtualisiert und oft über proprietäre Software administriert. Hier verwendet beispielsweise IONOS, der von der bitExpert AG gewählte Anbieter, den eigens entwickelten Software-Stack. Mit dem Data Center Designer (DCD) steht den Kunden eine komfortable Lösung zur Verfügung, um virtuelle Rechenzentren über eine grafische Oberfläche zu verwalten, wie in Abbildung 2.3 zu sehen. Anbieter wie AWS oder Microsoft Azure verfügen über ähnliche Lösungen.

Generell kann, wie in Abbildung 2.4 dargestellt, zwischen drei Arten von IaaS unterschieden werden, Public IaaS, Private IaaS und Hybrid IaaS (IONOS 2021a; Serrano, Gallardo und Hernantes 2015). Wie die Bezeichnung schon vermuten lässt, handelt es sich bei Public IaaS Angeboten um Dienstleistungen, die für jeden zu-

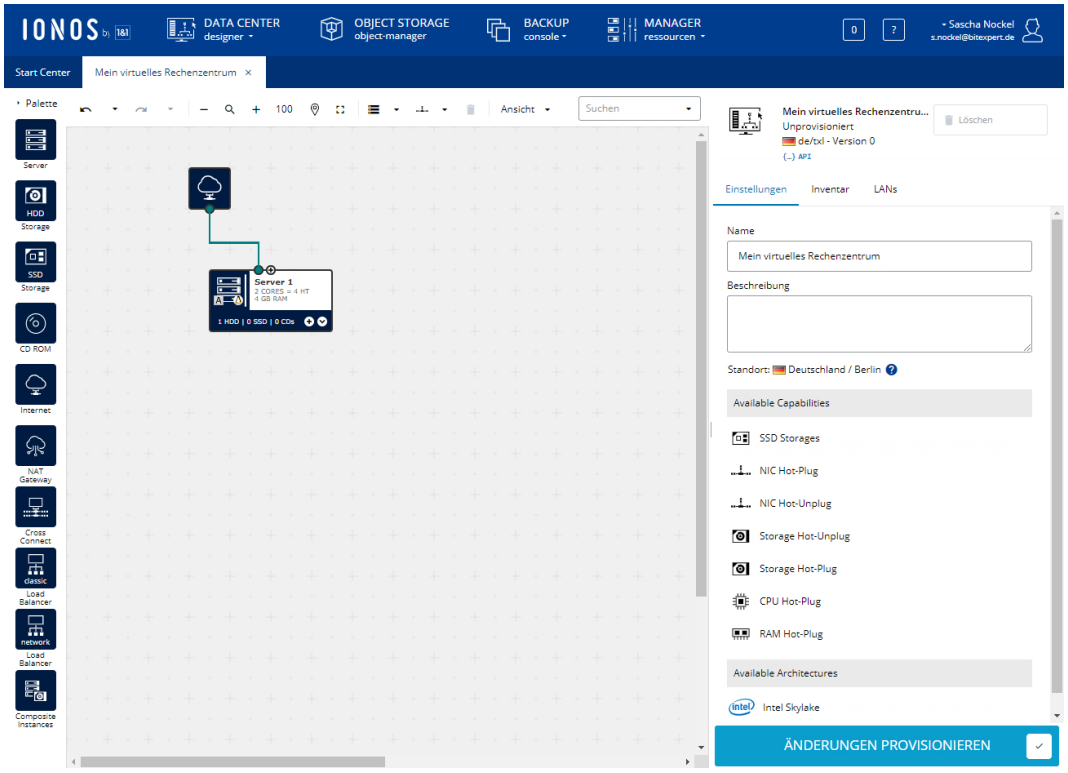

Abbildung 2.3: Screenshot der DCD Weboberfläche.

gänglich sind, Beispiele wären hier die schon genannten Anbieter IONOS, AWS und Azure.

Private IaaS Angebote werden dagegen in der Regel nur für einen Kunden betrieben. Dies kann der Kunde entweder selbst tun, oder einen externen Anbieter damit beauftragen, in seinem Rechenzentrum Hardware für die exklusive Nutzung durch den Kunden bereitzustellen. Hybrid IaaS ist eine Mischung der beiden zuvor genannten Arten, hier verwendet der Kunde zum Beispiel für die Speicherung und Verarbeitung von sensiblen Daten das eigene Rechenzentrum und für den Betrieb von weniger sicherheitskritischen oder kurzlebigen Anwendungen ein Public IaaS Angebot (Serrano, Gallardo und Hernantes 2015).

Da bei IaaS lediglich die Infrastruktur vom Anbieter verwaltet wird (mit Ausnahme von Private IaaS), fallen Aufgaben wie das Aufsetzen und Konfigurieren von Virtuellen Maschinen und Netzwerken sowie der Betrieb von Diensten und das Sorgen für Sicherheit, in den Aufgabenbereich der Kunden. Die garantierte Verfügbarkeit der Dienste sowie der Ressourcen, welche zur Verfügung stehen, legen die Anbieter in der Regel in ihren SLAs fest. Gegebenenfalls kann ein Anbieter auch verschiedene SLAs für verschiedene Produkte festlegen. So besteht zum Beispiel bei AWS

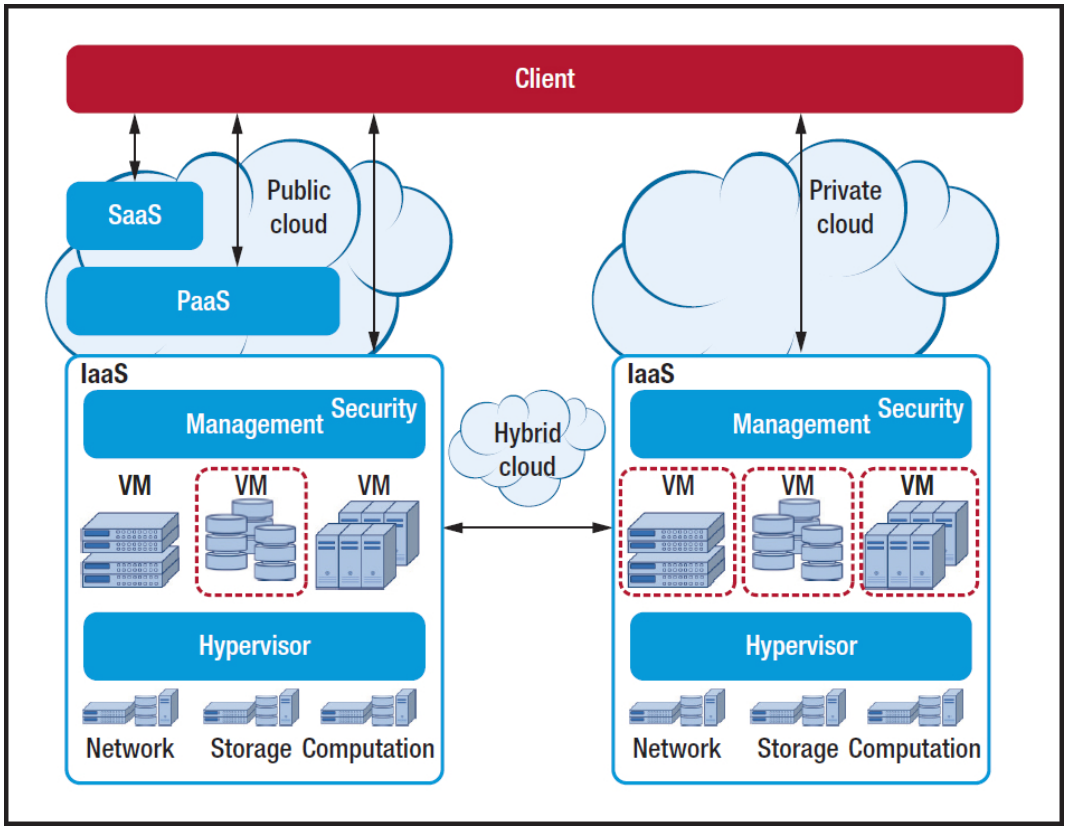

Abbildung 2.4: Die drei Cloud-Modelle. SaaS steht für Software as a Service, PaaS steht für Plattform as a Service (Serrano, Gallardo und Hernantes 2015).

oder Azure die Möglichkeit, mit wenig Aufwand Einstellungen für die Datenredundanz von verwendetem Speicherplatz festzulegen. Die gespeicherten Daten können so ohne größeres Zutun der Kunden per Knopfdruck auf die verschiedenen Rechenzentren der Anbieter repliziert werden, während sie für den Kunden selbst über eine einheitliche Schnittstelle zugänglich sind. Auch wenn hier einfache Lösungen für redundante Datenhaltung und hochverfügbare Dienste angeboten werden, schließen die SLAs der Anbieter eine Kompensation von finanziellen Schäden bei entstandenen Ausfallzeiten normalerweise aus (Serrano, Gallardo und Hernantes 2015).

Gerade deswegen sollte beim Schritt in die Cloud und besonders bei Verwendung von IaaS Diensten, immer berücksichtigt werden, dass mit Ausfällen zu rechnen ist (Vishwanath und Nagappan 2010). Dies sollte bei der Gestaltung der Cloud Infrastruktur berücksichtigt und ein umfassendes Backup Konzept für gespeicherte Daten angefertigt werden. Außerdem ist es empfehlenswert, die Infrastruktur schon von Anfang an ausfallsicher beziehungsweise hochverfügbar zu gestalten (Serrano, Gallardo und Hernantes 2015).

2.1.3 Infrastructure as Code

Cloud Computing ist in der heutigen Zeit in vielen Bereichen nicht mehr wegzudenken. Vorbei sind die Zeiten, in denen für jeden Dienst ein Server als Hardware provisioniert, manuell eingerichtet und ausgeliefert wurde. Applikationen sind längst nicht mehr abhängig von bestimmter Hardware und die Anforderungen an die Infrastruktur eines Unternehmens ändern sich ständig. Klassische Prozesse zur Verwaltung von Infrastruktur stoßen hier schnell an ihre Grenzen, da IT-Abteilungen aufgrund der Geschwindigkeit, mit der neue Server in der Cloud provisioniert werden können, schnell an die Grenzen dessen stoßen, was mit manueller Arbeit geleistet werden kann. Durch diese Geschwindigkeit nimmt die Anzahl der Systeme und die Komplexität der Infrastruktur ständig zu. Hier den Überblick zu wahren und alle Systeme in einem konsistenten Zustand zu halten, ist ohne Automatisierung schlicht nicht möglich (Morris 2016).

Moderne Infrastruktur muss flexibel sein, gleichzeitig dürfen Änderungen nicht zur Folge haben, dass andere Systeme nicht mehr funktionieren. Diese Flexibilität ist ohne automatisch reproduzierbare Systeme jedoch schlicht nicht zu erreichen, denn jeder Administrator hat eine eigene Herangehensweise und trifft andere Entscheidungen beim Provisionieren eines neuen Systems. Auch Dokumente und Richtlinien für solche Prozesse verringern die Flexibilität eher als sie zu verbessern, weil es immer Spezialfälle geben wird, die eine Änderung erfordern, welche nicht im Prozess erfasst ist (Morris 2016).

Um diese Probleme zu lösen und im gleichen Atemzug die Nachvollziehbarkeit und Reproduzierbarkeit von Infrastruktur zu gewährleisten, hat sich in den letzten Jahren der Begriff Infrastructure as Code (IaC) etabliert. Das Ziel von IaC ist es, Infrastruktur und Systeme nicht mehr manuell zu provisionieren und zu konfigurieren, sondern diese Arbeit mit Hilfe von verschiedenen Werkzeugen und vorher definierten Vorlagen und Skripts zu automatisieren. Diese Definitionen liegen als Code vor und können, wie herkömmlicher Quellcode auch, versioniert werden. Die Versionierung der Definitionen der Infrastruktur bringt diverse Vorteile mit sich. Zum einen sind Änderungen nachvollziehbar, da diese in der Regel zumindest mit einem Kommentar zur jeweiligen Änderung versehen werden. Zum anderen kann der Rest des Teams ebenfalls sehen, was geändert wurde, und gegebenenfalls Fehler aufzeigen, die dem ursprünglichen Autor nicht aufgefallen sind. Zusätzlich trägt dies dazu bei, dass alle Beteiligten stets wissen, was von wem, wann geändert wurde. Somit entfallen langwierige Prozesse zur Genehmigung von Änderungen, da

diese nun, wie bei der agilen Softwareentwicklung, über etablierte Methoden zur gemeinsamen Arbeit an Quellcode abgehandelt werden können (Morris 2016).

2.2 Automatisierungswerkzeuge

2.2.1 Terraform

Terraform ist ein IaC Werkzeug und wird von der Firma HashiCorp als Open Source Variante[2] unter der Mozilla Public License 2.0 sowie in einer Enterprise Version angeboten. Im Gegensatz zu Werkzeugen für das Konfigurationsmanagement wie Ansible (Unterabschnitt 2.2.2), ist Terraform darauf ausgelegt, Infrastruktur zu provisionieren und zu verwalten. Daher wird empfohlen, lediglich die Grundkonfigurationen mit Terraform vorzunehmen und die Infrastruktur so für den Betrieb vorzubereiten (HashiCorp 2021k).

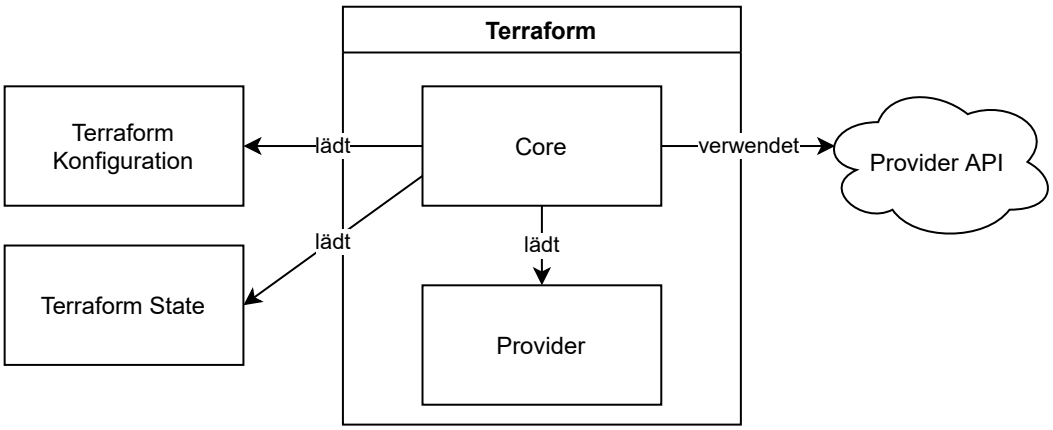

Abbildung 2.5: Diagramm zur Veranschaulichung der Terraform Architektur.

Die grundlegende Funktionsweise von Terraform besteht darin, mithilfe der sogenannten Provider, Plugins, welche von verschiedenen Anbietern entwickelt werden können, die Ressourcen der Infrastruktur mithilfe einer Terraform Konfiguration zu steuern, zu verwalten und zu ändern. Ein Provider kann zum Beispiel von einem IaaS Anbieter wie IONOS[3] zur Verfügung gestellt werden, um eine Schnittstelle zur Provisionierung von Infrastruktur zu bieten oder aber zur Konfiguration von Anwendungen wie GitLab[4] verwendet werden. Prinzipiell lässt sich alles steuern

[2] https://github.com/hashicorp/terraform
[3] https://registry.terraform.io/providers/ionos-cloud/ionoscloud/latest
[4] https://registry.terraform.io/providers/gitlabhq/gitlab/latest

und verwalten, wofür ein Provider existiert, wobei immer die Dokumentation zur Verwendung der jeweiligen Provider zu beachten ist (HashiCorp 2018).

Abbildung 2.5 veranschaulicht die Kapselung des Provider Application Programming Interface (API) durch Terraform Provider Plugins. So können zum Beispiel auch klassische Virtualisierungslösungen wie VMware vSphere über einen Terraform Provider[5] gesteuert werden.

Die fünf grundlegenden Terraform Befehle sind (HashiCorp 2021b):

1. init: Arbeitsverzeichnis für andere Befehle vorbereiten.
2. validate: Prüfen, ob die Terraform Konfiguration gültig ist.
3. plan: Änderungen anzeigen, die erforderlich sind, um die aktuelle Terraform Konfiguration umzusetzen.
4. apply: Infrastruktur erstellen und/oder aktualisieren.
5. destroy: Zerstörung der zuvor erstellten Infrastruktur.

Die Terraform Konfiguration definiert den gewünschten Zustand der Infrastruktur, die Befehle *plan* und *apply* zeigen, welche Änderungen erforderlich sind, und setzen diese um. Damit Terraform bei späteren Änderungen der Konfiguration nicht die gesamte Infrastruktur zerstört und neu aufbaut beziehungsweise diese doppelt provisioniert, wird der aktuelle Zustand nach jedem *apply* Befehl im Terraform State gespeichert. Diese Datei beinhaltet den gesamten Zustand der Infrastruktur mit allen vom Provider vergebenen IDs, damit Ressourcen bei späteren Änderungen der Konfiguration nicht komplett neu erstellt, sondern nach Möglichkeit referenziert und geändert werden können (HashiCorp 2018). Der Terraform State kann entweder lokal gespeichert oder in einem zentralen Speicher wie zum Beispiel dem Terraform Hypertext Transfer Protocol (HTTP) Backend von GitLab abgelegt werden (GitLab 2021b).

Eine Terraform Konfiguration zum Erstellen einer VM mit dem IONOS Provider könnte wie folgt aussehen:

```
# Benötigte Provider definieren
terraform {
  required_providers {
    ionoscloud = {
      source = "ionos-cloud/ionoscloud"
      version = "5.2.4"
    }
```

[5] https://registry.terraform.io/providers/hashicorp/vsphere/latest

```
  }
}

# Provider konfigurieren
provider "ionoscloud" {
  username = "username"
  password = "password"
}

# Virtuelles Rechenzentrum für zu provisionierende VM erstellen
resource "ionoscloud_datacenter" "example_dc" {
  name       = "Mein virtuelles Rechenzentrum"
  location   = "de/txl"
  description = "Beispiel Rechenzentrum, verwaltet mit Terraform"
}

# Netzwerk definieren
resource "ionoscloud_lan" "lan1" {
  datacenter_id = ionoscloud_datacenter.example_dc.id
  # public = true bedeutet LAN 1 ist an das Internet angebunden
  public = true
}

# VM konfigurieren
resource "ionoscloud_server" "example_vm" {
  name              = "example-vm"
  datacenter_id     = ionoscloud_datacenter.example_dc.id
  cores             = 1
  # 4096 bedeutet hier 4096MB
  ram               = 4096
  availability_zone = "AUTO"
  cpu_family        = "INTEL_SKYLAKE"
  image_password    = "root_password"
  image_name        = "Ubuntu-20.04-LTS-server-2021-07-01"

  volume {
    name      = "store01.example_vm"
    # 40 bedeutet hier 40GB
    size      = 40
    disk_type = "HDD"
  }

  # VM an LAN 1 anschließen und über DHCP automatisch öffentliche IP beziehen
  nic {
    lan             = ionoscloud_lan.lan1.id
    dhcp            = true
    firewall_active = false
  }
}
```

Listing 2.1: Beispiel für eine Terraform Konfiguration

Innerhalb einer Terraform Konfiguration können außerdem auch Ressourcen referenziert werden, wie in der *ionoscloud_lan* und *ionoscloud_server* Ressource in

Listing 2.1 zu sehen. So wird in diesem Beispiel Local Area Network (LAN) 1 dem erstellten virtuellen Rechenzentrum zugewiesen und das erstellte LAN wiederum der Netzwerkkarte des erstellten Servers. Auf diese Art und Weise können Ressourcen miteinander verknüpft werden, ohne dass die jeweiligen IDs zuvor bekannt sein müssen. Die IDs werden nach Erstellung der Ressourcen im Terraform State abgelegt. Es können in der Konfiguration aber weiterhin die symbolischen Verknüpfungen genutzt werden (HashiCorp 2018).

2.2.2 Ansible

Ansible ist ein Werkzeug zum Konfigurationsmanagement von verschiedensten Systemen. Solche Systeme können Server in der Cloud, lokale Server oder auch andere Hardware wie Router oder Switches sein (RedHat 2017). Das Open Source Projekt[6] ist unter der GNU General Public License v3.0 lizenziert und hat die Firma Red Hat als Hauptsponsor. Von Red Hat werden außerdem auch kommerzielle Produkte wie die Red Hat Ansible Automation Platform oder Red Hat Ansible Tower angeboten.

Im Gegensatz zu vielen anderen Lösungen zum Konfigurationsmanagement benötigt Ansible keinerlei Agenten oder sonstige Installationen auf den Zielsystemen, da die Verbindung zu den Hosts im Fall von Linux mit Secure Shell (SSH) und bei Windows Systemen mithilfe von Windows Remote Management (WinRM) hergestellt wird. Hierdurch kann Ansible dezentral verwendet werden, ohne eine separate Authentifizierung durchzuführen, da bestehende Rechtestrukturen und Konten genutzt werden können, um eine Verbindung zu den Zielsystemen herzustellen (RedHat 2017).

Abbildung 2.6 zeigt die Zusammenhänge der verschiedenen Ansible Komponenten. Die Konfiguration der Zielsysteme erfolgt mithilfe von sogenannten Playbooks, diese Playbooks werden in YAML Ain't Markup Language (YAML) Dateien definiert und können mehrere Plays (Gruppierungen von Tasks) mit verschiedenen Tasks beinhalten. Jeder Task definiert einen bestimmten Zustand, der auf den Zielsystemen hergestellt werden soll. Das kann zum Beispiel das Installieren eines bestimmten Softwarepakets, das Erstellen eines Ordners oder das Kopieren einer Datei auf die Zielsysteme sein. Tasks verwenden Modules zum Ausführen der nötigen Befehle auf den Zielsystemen. Diese Modules beinhalten den Code, der nötig ist, um

[6] https://github.com/ansible/ansible

den gewünschten Zustand auf den Zielsystemen herzustellen. Außerdem können Tasks auch in Roles zusammengefasst werden, um einen bestimmten Ablauf, zum Beispiel eine Folge von Tasks zum Installieren und Konfigurieren eines Webservers, wiederverwendbar zu machen. Zusätzlich zu Roles können auch Collections erstellt werden. Collections sind eine Sammlung von Modules, die gewisse Aufgaben erfüllen. Der Ansible Core beinhaltet bereits eine Vielzahl nützlicher Modules, es können jedoch auch von der Community erstellte Roles und Collections von der Ansible Galaxy Plattform[7] verwendet werden. Die Ansible Core Modules sind alle idempotent, ein mehrmaliges Ausführen ändert also nichts am Zielsystem, sofern der gewünschte Zustand schon vorliegt, generell sollten auch selbst entwickelte Modules idempotent gestaltet werden (RedHat 2017).

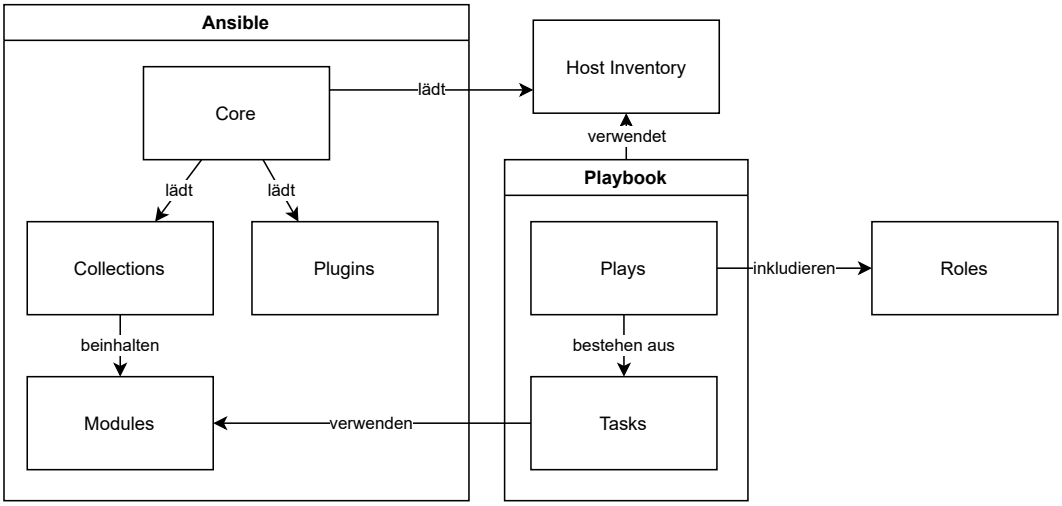

Abbildung 2.6: Diagramm zur Veranschaulichung der Ansible Architektur.

Die Hosts, auf welche ein Playbook angewendet werden soll, werden im sogenannten Host Inventory definiert. Dies ist eine Liste von IP-Adressen oder Hostnamen, auf die sich Ansible verbinden soll. So kann zum Beispiel parallel auf allen definierten Webservern das Webserver Playbook und auf allen Datenbankservern das Datenbank Playbook angewendet werden. Auf diese Art ist es möglich, gleichzeitig auf einer Vielzahl von Systemen Änderungen vorzunehmen sowie Software zu installieren und Aktualisierungen durchzuführen (RedHat 2017). Um ein erstelltes Playbook auf Korrektheit zu testen oder zu prüfen, welche Änderungen an den Zielsystemen vorgenommen werden würden, können außerdem die Kommandozeilenargumente *check* und *diff* verwendet werden. *check* zeigt an, welche Änderungen vorgenommen werden würden und *diff* liefert einen Vorher-Nachher-Vergleich der

[7] https://galaxy.ansible.com/

Änderungen. Die beiden Argumente können auch in Kombination verwendet werden, um anzuzeigen was geändert werden wird und wie zum Beispiel eine Konfigurationsdatei vor und nach dem Anwenden des Playbooks aussehen würde (Ansible 2021).

Ein Ansible Playbook mit zugehörigem Host Inventory könnte wie folgt aussehen:

```
[webserver]
123.123.123.123
my.example.com
```

Listing 2.2: Beispiel für ein Ansible Host Inventory.

```
# Host Gruppe für Playbook definieren
- hosts: webserver
  # Werde Root Benutzer, sobald die Verbindung hergestellt wurde
  become: true

  tasks:
    - name: Webserver und Reverse Proxy installieren
      ansible.builtin.apt:
        # Pakete definieren, welche installiert werden sollen
        name:
          - apache2
          - nginx
        # state: present bedeutet, installiere die Pakete nur, falls sie noch nicht
            vorhanden sind
        state: present
        # Erneuere die Paketliste vor dem installieren
        update_cache: yes
```

Listing 2.3: Beispiel für ein Ansible Playbook.

Nach dem Anwenden des Playbooks (Listing 2.2) auf das Host Inventory (Listing 2.3) sind auf den beiden Hosts der Apache Webserver und der NGINX Reverse Proxy installiert. Bei erneutem Abspielen des Playbooks, zum Beispiel wenn noch zusätzliche Pakete hinzugefügt wurden, werden lediglich die Pakete installiert, welche noch nicht vorhanden sind.

2.3 Continuous Integration und Continuous Delivery

Continuous Integration (CI) und Continuous Delivery (CD) sind Begriffe, welche oft mit moderner und agiler Softwareentwicklung in Verbindung gebracht werden. Während sich CI vor allem damit beschäftigt, die Komponenten der entwickelten Software bei jeder Änderung zu integrieren und zu testen, beschäftigt sich CD vor allem damit, die Software in einen lauffähigen Zustand zu bringen, indem der Co-

de beispielsweise kompiliert wird, um eine ausführbare Datei zu generieren. CD wird daher oft synonym für Continuous Deployment verwendet, weil die beiden Begriffe zwar etwas ähnliches aber nicht das gleiche meinen. Continuous Deployment unterscheidet sich von Continuous Delivery darin, dass die lauffähige Software zusätzlich vollautomatisch an die Zielsysteme ausgeliefert wird. Eine Continuous Deployment Pipeline kann durchaus mehrere Stufen haben, zum Beispiel kann die Anwendung zuerst auf Test oder Stagingsysteme ausgeliefert werden und sobald die Funktion hier sichergestellt wurde ebenfalls auf die Produktivsysteme (Humble und Farley 2010).

Die Betonung bei allen drei Begriffen liegt auf *Continuous*, denn nutzbringend und effizient wird eine CI/CD Pipeline erst dann, wenn die Software auf Knopfdruck mehrmals am Tag oder sogar mehrmals pro Stunde vollautomatisch integriert, getestet und ausgeliefert werden kann. Durch diese Automatisierung und die ständig vorliegende lauffähige Software, können Fehler schon sehr früh im Entwicklungsprozess erkannt und behoben werden. Außerdem werden Fehler, welche bei manueller Auslieferung von Software auftreten können, vermieden. Diese Prozesse sind per Knopfdruck reproduzierbar. Dadurch gestaltet sich ein Zurückrollen auf die letzte lauffähige Version ebenso einfach wie das Ausliefern der neuesten (Humble und Farley 2010).

Diesen Techniken und Prinzipien liegt immer ein System zur Versionsverwaltung des Quellcodes sowie ein System zum Ausführen von derartigen Pipelines zugrunde (Humble und Farley 2010). Prominente Beispiele sind hier vor allem GitLab[8] und Jenkins[9], wobei GitLab noch einen Schritt weiter geht als Jenkins, da es nicht nur die Ablaufumgebung für CI/CD Pipelines, sondern ebenfalls Git Repositories, Wikis, Issue Tracking, Pull und Merge Requests, verschiedene Werkzeuge für das Projektmanagement sowie viele weitere Funktionen bietet und damit eine komplette DevOps (Development und Operations) Plattform bietet (GitLab 2021a).

Abbildung 2.7 zeigt ein Beispiel für eine in GitLab erstellte Continuous Deployment Pipeline unter Verwendung von Terraform (Unterabschnitt 2.2.1) und Ansible (Unterabschnitt 2.2.2). Zu sehen sind die verschiedenen Stages der Pipeline sowie deren jeweilige Jobs. Zwischen verschiedenen Stages und Jobs können außerdem Artefakte wie zum Beispiel ein von Terraform für Ansible erstelltes Host Inventory ausgetauscht werden. Eine Pipeline wird in der Regel immer an einen bestimm-

[8] https://about.gitlab.com/
[9] https://www.jenkins.io/

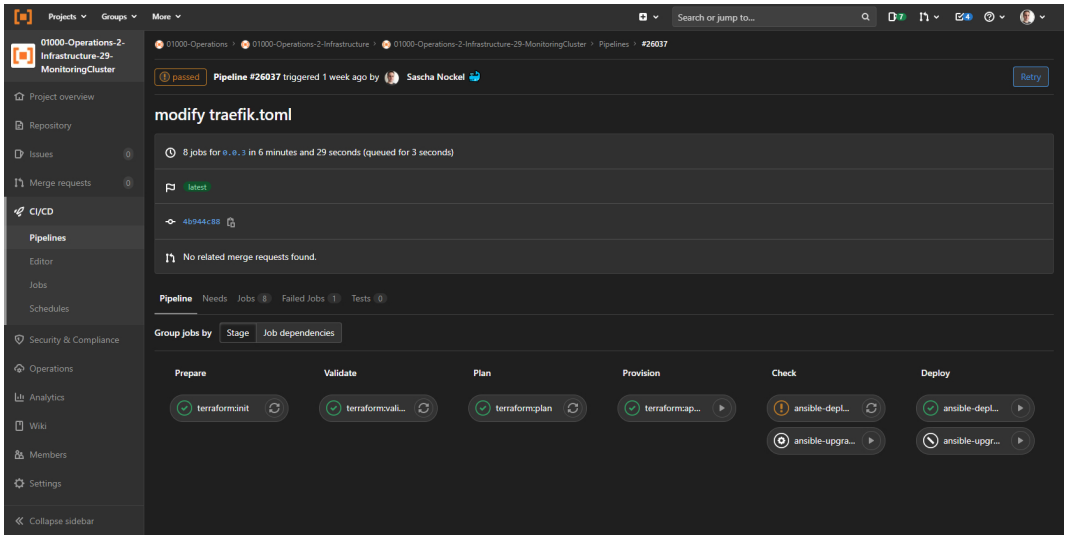

Abbildung 2.7: Screenshot einer GitLab Deployment Pipeline unter Verwendung von Terraform und Ansible.

ten Commit oder Tag gebunden, um in der Pipeline mit dem jeweiligen Stand des Quellcodes arbeiten zu können.

2.4 HashiCorp Technologie Stack

In diesem Kapitel werden drei für den Betrieb der bitExpert Cloud essenzielle Softwareprodukte vorgestellt. Dazu gehören Consul, Vault und Nomad.

Consul bildet die Basis für die Vernetzung der Dienste und die Kommunikation innerhalb des Clusters. Hier werden die verschiedenen von Nomad orchestrierten Dienste mit ihren IPs und Ports registriert, um diese über eine zentrale Service Registry auffindbar zu machen.

Vault hingegen stellt einen Secret Store innerhalb des Clusters zur Verfügung. So können über Nomad oder auch aus CI/CD Pipelines heraus, verschiedene Secrets wie zum Beispiel Passwörter oder Zertifikate abgefragt werden.

Nomad verbindet sich nun mit den APIs von Consul und Vault und ermöglicht so innerhalb der Nomad Jobs das dynamische Abrufen der Adressen anderer Dienste im Cluster sowie von Secrets aus Vault. So können innerhalb der Nomad Jobs transparent alle Möglichkeiten von Consul und Vault genutzt werden, um Dienste zu konfigurieren, zu registrieren und anzusprechen.

2.4.1 Consul

Consul ist eine Service Mesh Lösung von HashiCorp und wird als Open Source Variante[10] unter der Mozilla Public License 2.0 sowie in einer Enterprise Version angeboten und bietet folgende Kernfunktionalitäten (HashiCorp 2021e):

- Service Discovery
- Health Checking
- Key/Value Store
- Sichere Service Kommunikation
- Multi Datacenter fähig

Im Wesentlichen können, zum Beispiel über Nomad, automatisch Services bei Consul registriert werden. Diese Services wiederum beinhalten Informationen wie den aktuellen Status des Services, welcher mithilfe von Health Checks ermittelt wird, sowie die IP-Adresse der Node auf welcher der Service momentan ausgeführt wird. Außerdem wird auch der dynamische Port des Services hinterlegt, da sich dieser je nach Verteilung der Services ändern kann (HashiCorp 2021e). Mit Consul Connect kann nun mithilfe von Sidecar Proxies ein Service Mesh aufgebaut werden. Im Prinzip erhält jeder Service wie in Abbildung 2.8 dargestellt einen eigenen Sidecar Proxy. Diese Sidecar Proxies werden von Consul gesteuert und können entweder untereinander oder über Rechenzentren hinweg verschlüsselt miteinander kommunizieren. Die Vorteile eines solchen Service Meshs liegen vor allem darin, dass Services nicht mehr verschiedenste Technologien und Logiken implementieren müssen, um mit anderen verteilten Services kommunizieren zu können. Da die Kommunikation außerdem vorrangig unter den Sidecar Proxies stattfindet, müssen Applikationen keinerlei Kenntnis über ihre Umgebung haben, um mit Services in einem anderen Rechenzentrum kommunizieren zu können (HashiCorp 2021d; RedHat 2021).

[10] https://github.com/hashicorp/consul

Abbildung 2.8: Kommunikation zweier Services über Consul Connect Sidecar Proxies (HashiCorp 2021m).

Generell wird Consul als verteilter Dienst betrieben. In einem Consul Cluster laufen gewöhnlich mehrere Server und Clients, wobei es sich empfiehlt, entweder drei oder fünf Consul Server zu betreiben, da die Datenhaltung mithilfe des Raft Consensus Protokolls auf den Servern erfolgt und immer ein Leader für das Cluster bestimmt wird (Ongaro und Ousterhout 2014; HashiCorp 2021c). Die Consul Server übernehmen die Aufgaben zur Verwaltung des Consul Clusters, dazu gehört das Weiterleiten von Anfragen an den Leader des Clusters und das Beantworten von Anfragen der Clients. Die Clients eines Consul Clusters rufen Informationen zu registrierten Services ab. Die Ergebnisse dieser Anfragen werden außerdem zur Beschleunigung wiederholter Anfragen auf den jeweiligen Clients zwischengespeichert, ähnlich wie dies bei Domain Name System (DNS) Anfragen der Fall ist. Consul ist außerdem darauf ausgelegt, ein Service Mesh über mehrere Rechenzentren hinweg aufzubauen. Daher kommunizieren Consul Server in anderen Rechenzentren oder Regionen wie in Abbildung 2.9 zu sehen mithilfe eines Gossip Protokolls über das Wide Area Network (WAN). So können zum Beispiel Anfragen an *Datacenter 1*, welche für *Datacenter 2* bestimmt sind, unter den Consul Servern über Remote Procedure Calls (RPC) weitergeleitet werden. Der Leader eines Rechenzentrums ist dafür verantwortlich, alle Anfragen und Transaktionen zu bearbeiten. Die anderen Server, welche im Follower-Modus agieren, sowie die Agenten auf den verschiedenen Nodes, welche als Clients agieren, leiten alle an sie gestellten Anfragen an den Leader weiter. Im LAN eines Consul Clusters wird ebenfalls ein Gossip Protokoll verwendet, um Nachrichten unter den Servern und Clients im lokalen Netz auszutauschen und gegebenenfalls auf wichtige Ereignisse wie die Bestimmung eines neuen Leaders hinzuweisen (HashiCorp 2021c).

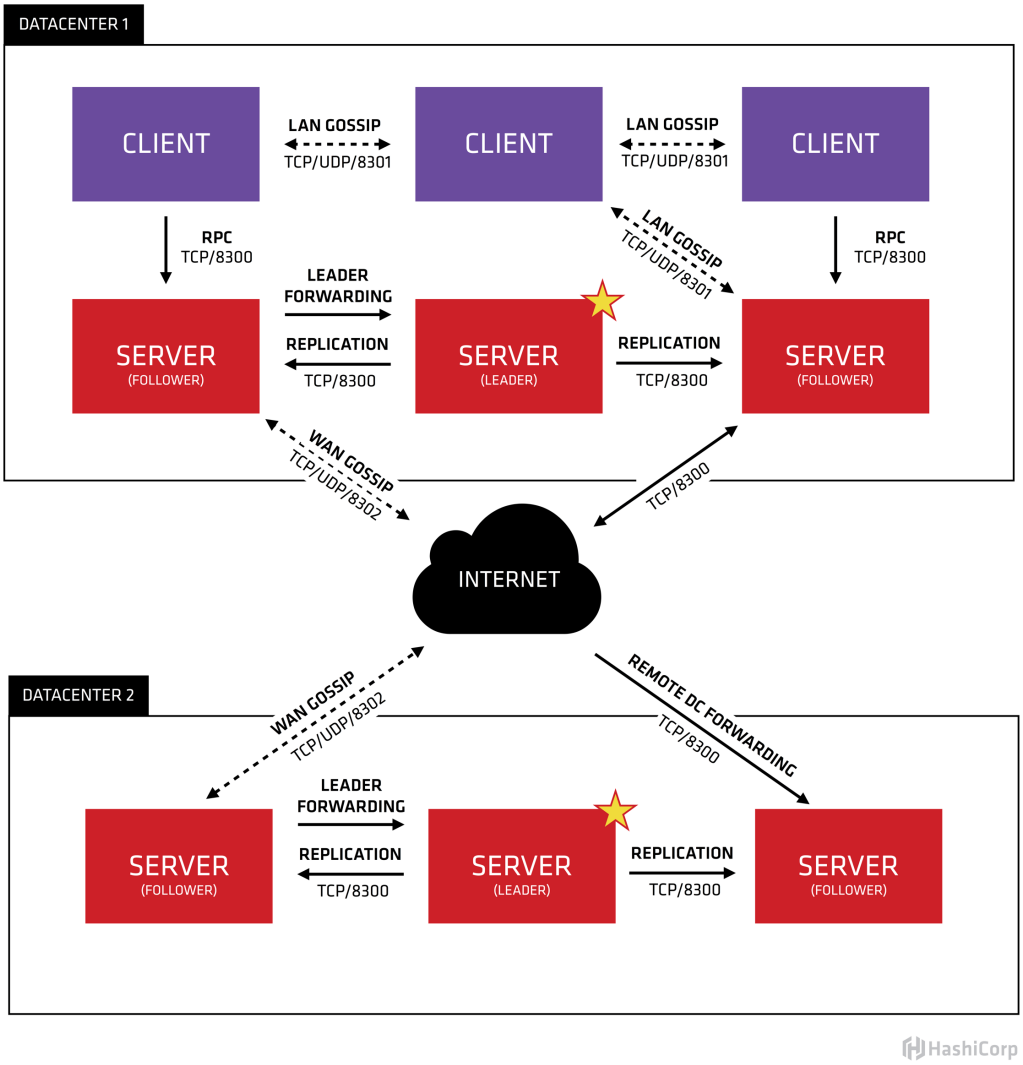

Abbildung 2.9: Auf zwei Rechenzentren verteiltes Consul Cluster (HashiCorp 2021c).

2.4.2 Vault

Vault ist als kostenlose Open Source Variante[11] unter der Mozilla Public License 2.0 oder als Enterprise Version erhältlich. Vault ist im einfachsten Sinn ein Secret Store, es können zum Beispiel Passwörter, Zertifikate oder auch ganze Logins sowie Konfigurationen gespeichert werden. Durch die Architektur von Vault werden die gespeicherten Daten nie im Klartext auf die Festplatte geschrieben, sondern immer nur im verschlüsselten Zustand abgelegt. Dadurch sind die Daten selbst bei einer Kompromittierung des Systems nicht einsehbar, ohne den Master Key zu kennen (HashiCorp 2021f).

[11] https://github.com/hashicorp/vault

Folgende Kernfunktionen werden von Vault angeboten (HashiCorp 2021f):

- Sicherer Secret Store für Key/Value Paare
- Erstellung Dynamischer Secrets zum Beispiel für automatisierte Skripts
- Verschlüsselung aller Daten vor dem Speichern
- Leasing und Renewal von Secrets
- Widerruf von Secrets

Um Vault in einem HA Setup zu betreiben, werden externe Speicherlösungen zur Datenhaltung verwendet, anstatt dies über das Vault Cluster zu realisieren. Mögliche Speicherlösungen sind unter anderem Consul und Simple Storage Service (S3). Die Vault Nodes, welche zusammen ein Cluster bilden, agieren jedoch nicht zusammen. Stattdessen erstellt immer eine Node ein Lock in der Speicherlösung, um den anderen Nodes mitzuteilen, dass sie aktuell als Leader agiert. Alle anderen Nodes agieren von da an im Standby Modus und leiten alle an sie gestellten Anfragen an die Leader Node weiter. Sollte die Leader Node ausfallen, greift sich eine der Standby Nodes das Lock und agiert von da an als neuer Leader. Somit ist die Skalierbarkeit von Vault hauptsächlich von der zugrunde liegenden Speicherlösung abhängig (HashiCorp 2021o).

Unter anderem lässt sich Vault auch an Nomad anbinden, um bei neuen oder geänderten Nomad Jobs automatisch Secrets, wie zum Beispiel Datenbanklogins, abfragen und diese den Containern mitgeben zu können (HashiCorp 2021n).

2.4.3 Nomad

Nomad ist eine Alternative zum schwergewichtigen Kubernetes, das von Firmen wie Google und RedHat entwickelt wird (HashiCorp 2021h). Die Orchestrierungslösung ist als Open Source und als Enterprise Version erhältlich, die Open Source Variante[12] unterliegt der Mozilla Public License 2.0.

Ganz der Unix Philosophie “Do One Thing and Do It Well” folgend, legt Nomad im Gegensatz zu Kubernetes den Hauptfokus auf das Cluster Management und Scheduling, anstatt zu versuchen alle Bereiche einer Lösung zur Cluster Orchestrierung abzudecken. Ein Nomad Cluster kann aus einem einzigen Client bestehen, der gleich-

[12] https://github.com/hashicorp/nomad

zeitig als Server agiert, oder aber bis zu mehrere tausend Clients beinhalten und bis zu 2 Millionen Container verwalten (HashiCorp 2021l). Dies wird vor allem durch die leichtgewichtige Architektur erreicht. Denn um ein Nomad Cluster in Betrieb zu nehmen, ist für jeden Server und jeden Client lediglich eine ausführbare Datei und eine Konfigurationsdatei nötig (HashiCorp 2021j). Funktionalitäten wie Service Discovery und Secret Management wurden in andere HashiCorp Produkte wie Consul (Unterabschnitt 2.4.1) und Vault (Unterabschnitt 2.4.2) ausgelagert. Diese lassen sich jedoch nahtlos an Nomad anbinden (HashiCorp 2021a).

Neben der einfachen Administrierung und Inbetriebnahme eines Nomad Clusters lassen sich mit den sogenannten Nomad Jobs über eine deklarative Spezifikation der Dienste, welche betrieben werden sollen, relativ einfach komplexe Container-Workloads realisieren. Nomad Jobs beinhalten Tasks und Task Gruppen, jeder Task muss zu einer Task Gruppe gehören und jede Task Gruppe wird nach Möglichkeit auf einen eigenen Client verteilt. Task Gruppen werden als sogenannte Allokationen auf die verfügbaren Nomad Clients verteilt, bei fehlerhaften Allokationen wird der Job automatisch auf den letzten funktionalen Stand zurückgesetzt. Andere Dienste im Cluster lassen sich über die Consul Service Discovery leicht und bei Bedarf verschlüsselt ansprechen. Außerdem bietet Nomad bereits einige Updatestrategien wie zum Beispiel Rolling Updates, um einen Container auf die neueste Version zu aktualisieren, ohne dass der Dienst offline geht (HashiCorp 2021j).

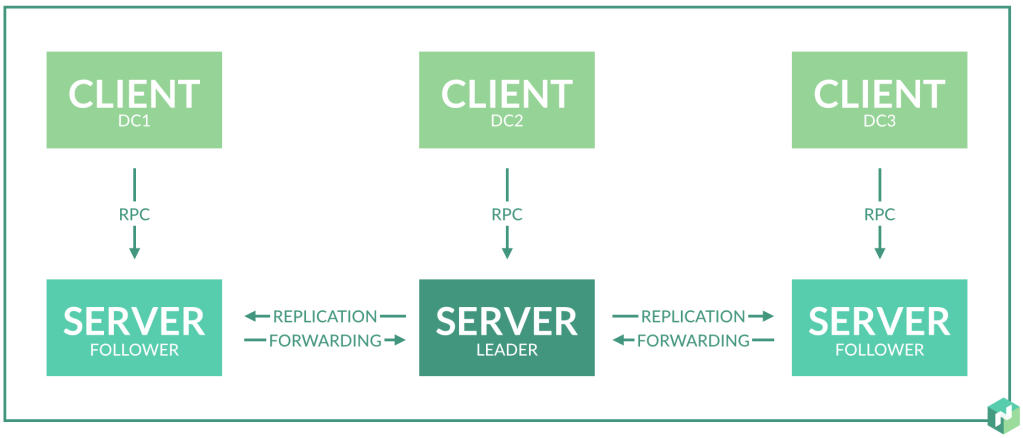

Abbildung 2.10: Einzelne Nomad Region mit Clients in verschiedenen Rechenzentren (HashiCorp 2021g).

Neben Linux Containern lassen sich auch Windows Container sowie Java Applikationen oder Batch Jobs über das Cluster betreiben, dies wird durch die Erweiterbarkeit mit Plugins ermöglicht (HashiCorp 2021i). Über Plugins lässt sich auch spezi-

elle Hardware wie Grafikkarten oder ein Cluster übergreifendes Speichersystem für zustandsbehaftete Anwendungen einbinden (HashiCorp 2021j).

Prinzipiell ist Nomad darauf ausgelegt eine Region mit mehreren Rechenzentren zu orchestrieren. Eine solche Nomad Region beinhaltet immer Server und Clients, wie in Abbildung 2.10 dargestellt. Hierbei kann jede Region Clients in mehreren verschiedenen Rechenzentren beinhalten, wobei sich die Server nicht zwangsweise im gleichen Rechenzentrum wie die Clients befinden müssen. Es muss lediglich die RPC Kommunikation über das Netzwerk sichergestellt sein (HashiCorp 2021g).

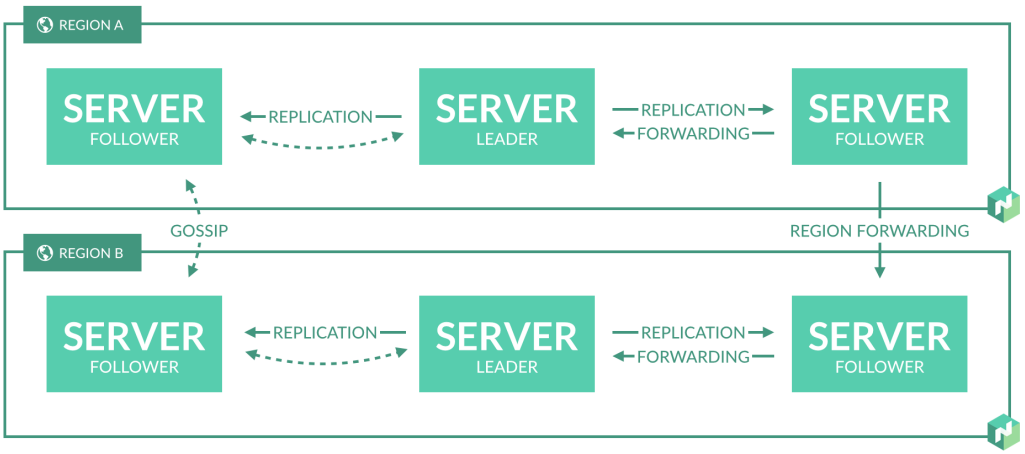

Abbildung 2.11: Zwei lose gekoppelte Nomad Regionen (HashiCorp 2021g).

Um eine höhere Ausfallsicherheit zu erzielen oder Georedundanz zu erreichen, können auch mehrere Nomad Regionen lose miteinander gekoppelt werden wie in Abbildung 2.11 zu sehen. Die Regionen sind unabhängig voneinander und teilen weder Jobs, Clients noch Zustand, auch werden keinerlei Daten zwischen den Regionen repliziert. Die Kopplung erfolgt mithilfe der Serf Bibliothek[13], welche ein Gossip Protokoll nach Das, Gupta und Motivala 2002 implementiert. Einfach gesagt funktioniert ein Gossip Protokoll ähnlich wie eine Infektionskrankheit. Sobald ein Rechner eine neue Nachricht erhält, verbreitet er sie an alle Rechner in seiner Umgebung, diese wiederum verteilen die Nachricht ebenfalls an alle Rechner in ihrer Umgebung, dies wird so lange fortgeführt, bis alle Rechner die Nachricht erhalten haben. Praktisch wird diese Verbreitung über Peer-to-Peer Kommunikation und mithilfe von Broadcasts im Netzwerk verwirklicht (Birman 2007).

Die Kopplung mehrerer Regionen ermöglicht es nun, Jobs an jede beliebige Region beziehungsweise jedes Rechenzentrum zu verteilen, ohne die jeweils verantwortli-

[13] https://www.serf.io/

chen Nomad Server direkt ansprechen zu müssen. Der angesprochene Nomad Server leitet Anfragen an ihn, welche für eine andere Region bestimmt sind, an die Nomad Server der Zielregion weiter. Außerdem kann so über jeden Nomad Server der Status aller Regionen und Rechenzentren abgefragt werden (HashiCorp 2021g).

Die Nomad Server, welche die Clients steuern, speichern den Zustand des Clusters mithilfe einer Raft Consensus Implementierung. Server einer Region bilden eine Consensus Gruppe, in der jeweils ein Leader bestimmt wird. Alle anderen Server besetzen die Rolle eines Followers und leiten Anfragen, die an sie gestellt werden, an den Leader weiter. Aufgrund der Funktionsweise des Raft Consensus Algorithmus sollten stets mindestens drei oder fünf Nomad Server für jede Region betrieben werden, um Datenverluste beim Ausfall eines Servers zu verhindern und das Cluster weiter steuern zu können. Eine höhere Anzahl Server führt lediglich zur Verlangsamung des Raft Consensus Algorithmus. Dies liegt darin begründet, dass für die Bestimmung eines neuen Leaders, welcher alle Anfragen entgegennimmt, sowie Transaktionen durchführt und das Cluster steuert, immer eine absolute Mehrheit notwendig ist. Diese bei der Abstimmung unter den Servern gebildete absolute Mehrheit wird auch Quorum genannt (Ongaro und Ousterhout 2014; HashiCorp 2021g).

2.5 Traefik Proxy Enterprise Edition

Traefik ist ein Reverse Proxy und Lastverteiler mit Fokus auf Cloud Umgebungen und Microservices. Im Gegensatz zur kostenlosen Traefik Proxy Variante[14], welche alle Funktionen in einem Paket anbietet, teilt Traefik EE die Verantwortlichkeiten in zwei Schichten oder auch Planes auf und wird grundsätzlich als verteilter Dienst betrieben (Containous 2021f).

Die Control Plane (Controller in Abbildung 2.12) ist von außen nicht erreichbar und verantwortlich für das Speichern von Daten wie Ereignissen oder Transport Layer Security (TLS) Zertifikaten. Außerdem werden die Verbindung zur Orchestrierungslösung (Orchestrator API in Abbildung 2.12), zum Beispiel Kubernetes, Nomad oder Consul (Unterabschnitt 2.4.1), sowie die Konfigurationen der Data Plane (Ingress Proxies in Abbildung 2.12) von der Control Plane gesteuert. Traefik bietet hier einen großen Vorteil für Cloud Umgebungen, da die verschiedenen Kon-

[14] https://github.com/traefik/traefik

figurationen für die jeweiligen Services hinter dem Proxy dynamisch über die APIs der Orchestrierungslösungen geladen werden können. Im Fall von Consul werden den Services Tags zugewiesen, anhand derer Traefik die entsprechenden Konfigurationen vornimmt. Diese Konfigurationen werden dynamisch geladen und es wird kein Neustart des Proxies oder ein erneutes Laden der Konfigurationsdatei benötigt. Die Daten der Control Plane werden über eine Raft Consensus Implementierung auf die verschiedenen Controller verteilt (Containous 2021e; Ongaro und Ousterhout 2014). Die Kommunikation unter den beiden Schichten findet außerdem ausschließlich verschlüsselt statt (Containous 2021f).

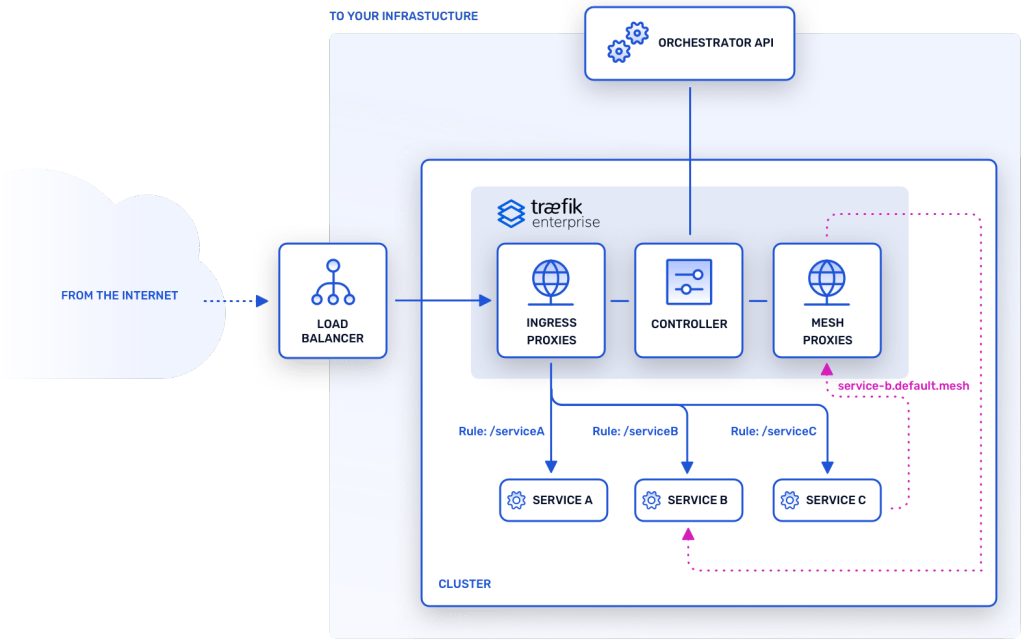

Abbildung 2.12: Traefik EE Architektur (Containous 2021f).

Abbildung 2.12 zeigt die Traefik EE Architektur, wobei hier nicht auf die Mesh Proxies Komponente eingegangen wird, da diese nur in einem Kubernetes Cluster verfügbar ist, im Prinzip aber dieselben Funktionalitäten wie Consul Connect (Unterabschnitt 2.4.1) bietet (Containous 2021d). In der Abbildung ist zu sehen, dass Anfragen aus dem Internet über einen Lastverteiler an die verschiedenen Ingress Proxies verteilt werden. Ein separater Lastverteiler vor den Ingress Proxies ist in dieser Architektur nötig, damit Traefik die interne Lastverteilung auf die verschiedenen Services übernehmen kann. In der kostenlosen Version des Traefik Proxies ist kein vorgeschalteter Lastverteiler nötig, da die Control und Data Planes hier nicht

getrennt werden, beziehungsweise nur eine Instanz für die Verarbeitung von Anfragen zuständig ist. Die Data Plane des Traefik EE Proxies ist zuständig für das Routing der Anfragen zwischen den verschiedenen Services. Beide Schichten sind außerdem horizontal skalierbar und können als HA Setup betrieben werden (Containous 2021e).

Für die Administrierung der beiden Schichten wird das Kommandozeilenwerkzeug *teectl* mitgeliefert, womit die Schichten gesteuert werden können, ohne separat auf die einzelnen Nodes zugreifen zu müssen (Containous 2021e).

2.6 Simple Storage Service

S3 ist eine von Amazon entwickelte Objektspeicherlösung, die sich mittlerweile als inoffizieller Standard für das Speichern von großen Datenmengen in der Cloud etabliert hat. Amazon selbst garantiert beim eigenen S3 Dienst eine Datenhaltbarkeit von 99,999999999% und bietet verschiedene Möglichkeiten für die Replikation der gespeicherten Daten an (Amazon 2021b).

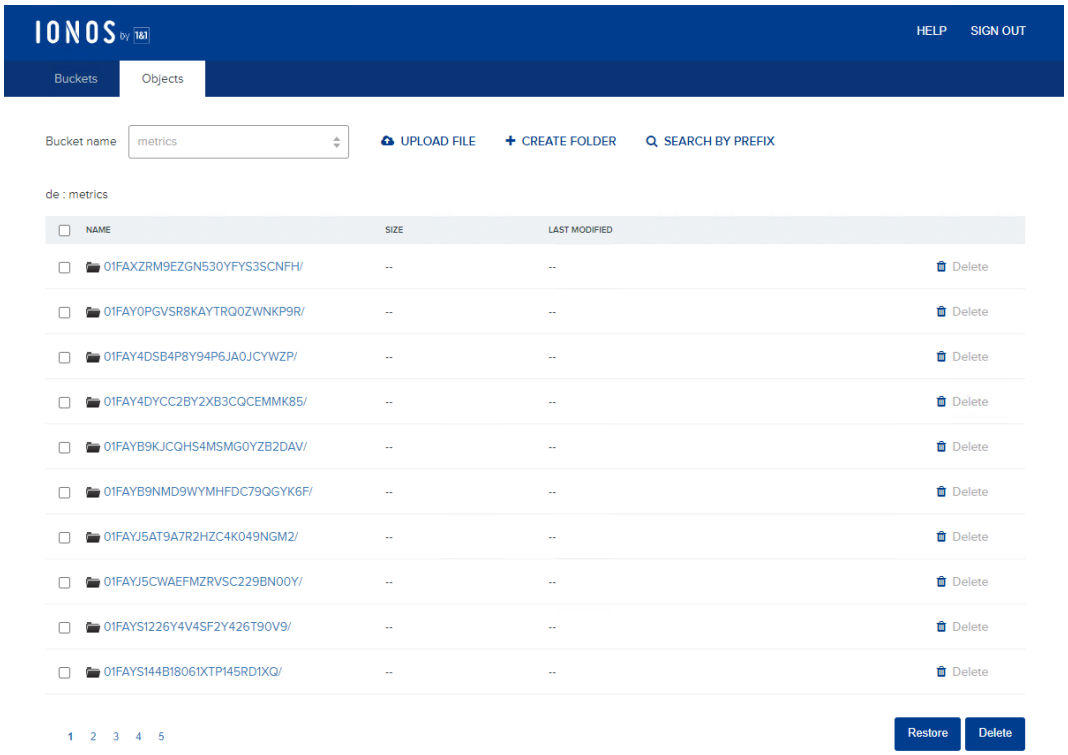

Abbildung 2.13: Screenshot der Weboberfläche des IONOS S3.

Das Grundprinzip von S3 ist das Ablegen der Nutzdaten als Objekte in den sogenannten Buckets. Diese Daten können dann über eine Representational State Transfer (REST) API abgerufen, bearbeitet und gelöscht werden (Amazon 2021a). Ein Bucket stellt praktisch nichts anderes als einen Ordner im S3 Speicher dar. In einen Bucket können dann wiederum beliebige Dateien hochgeladen und Verzeichnisse angelegt werden. Abbildung 2.13 zeigt den *metrics* Bucket und dessen Inhalt im IONOS S3 Speicher. Die zu sehenden Ordner sind das Ergebnis eines ersten Testlaufs mit Prometheus (Unterabschnitt 2.7.1) und Thanos (Unterabschnitt 2.7.2). Aufgrund der weiten Verbreitung von S3 bieten auch andere Anbieter, wie im Beispiel hier IONOS, eigene S3 Lösungen an, welche in der Regel dieselben APIs wie Amazons S3 implementieren beziehungsweise anbieten, um die Kompatibilität mit bestehenden Anwendungen zu gewährleisten (IONOS 2021b).

Außerdem besteht neben den kommerziellen S3 Angeboten auch die Möglichkeit, einen eigenen S3 Speicher zu betreiben. Hierfür existieren Open Source Implementierungen wie MinIO[15], welche ebenfalls die Amazon S3 APIs implementieren.

2.7 Monitoring Komponenten

2.7.1 Prometheus

Prometheus ist ein unter der Apache License 2.0 lizenziertes Open Source Projekt[16] welches von der Cloud Native Computing Foundation entwickelt wird. Die Hauptfunktionalität besteht aus dem Einsammeln von Metriken von verschiedensten Systemen sowie dem Abfragen dieser über die eigens entwickelte Abfragesprache PromQL. Im Kern besteht Prometheus aus drei Hauptkomponenten, dem Data Retrieval Worker, welcher für das Einsammeln der Metriken zuständig ist, einer Time Series Database (TSDB) und dem Web Frontend der Prometheus Instanz, über das PromQL Abfragen abgesetzt und Betriebsdaten ausgelesen werden können (Prometheus 2021i).

Prometheus folgt beim Einsammeln der Metriken grundsätzlich dem Pull-Prinzip. Das bedeutet, dass auf den sogenannten Targets keinerlei Daemon Prozess installiert werden muss, der Metriken an den Prometheus Server sendet. Stattdessen werden die Targets eigenständig vom Server abgefragt. Die Targets können entweder sta-

[15] https://min.io/
[16] https://github.com/prometheus/prometheus

tisch konfiguriert oder über Service Discovery Dienste wie zum Beispiel Consul (Unterabschnitt 2.4.1) dynamisch hinzugefügt werden. Diese Methode der Datenabfrage wurde bewusst gewählt, um beispielsweise Ausfälle von Diensten einfacher erkennen zu können sowie das Abfragen von Targets zu vereinfachen. Da die Abfrage der Metriken ausschließlich über HTTP stattfindet, müssen Targets lediglich einen Endpunkt zur Abfrage anbieten. Standardmäßig erwartet Prometheus Metriken unter dem */metrics* Endpunkt eines Targets. Es lassen sich jedoch auch andere Endpunkte definieren, welche das gleiche Datenschema implementieren (Prometheus 2021i).

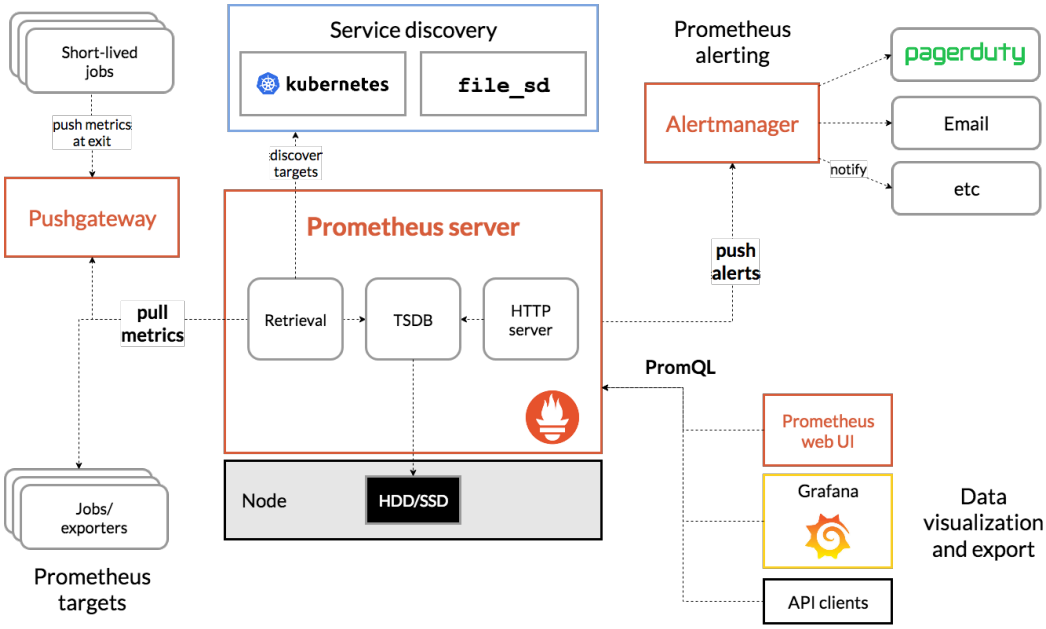

Abbildung 2.14: Exemplarisches Prometheus Setup mit Pushgateway, Alertmanager und Grafana (Prometheus 2021i).

Für Prozesse, die zu kurzlebig sind um effektiv über das Pull-Prinzip abgefragt werden zu können, bietet Prometheus die Möglichkeit ein Pushgateway (Abbildung 2.14 links oben) zu betreiben, an das die Metriken der kurzlebigen Prozesse gesendet werden können. Von diesem Pushgateway wiederum fragt der Prometheus Server dann über das Pull-Prinzip die gesammelten Metriken ab (Prometheus 2021i).

Für Dienste oder etwa Hardware, die keinerlei Prometheus Endpunkt implementieren, existieren außerdem offizielle sowie inoffizielle Client Libraries und die sogenannten Exporter. Offiziell unterstützt werden die Client Libraries für Go, Java, Scala, Python und Ruby. Jedoch existieren noch weitere von der Community entwi-

ckelte Erweiterungen (Prometheus 2021c). Ähnlich sieht es bei den Exportern aus (Abbildung 2.14 links unten). So bietet Prometheus beispielsweise offizielle Exporter für MySQL Server oder Linux Betriebssysteme, um Metriken von Diensten abfragen zu können, welche keinen eigenen Endpunkt implementieren. In einem solchen Szenario sammelt der Exporter relevante Informationen über zum Beispiel den MySQL Server, bringt diese in das richtige Format für Prometheus und stellt einen Endpunkt zur Abfrage bereit. Auch hier gibt es zahlreiche von der Community entwickelte Pakete, um zum Beispiel Metriken von GitHub Repositories abfragen zu können (Prometheus 2021e).

Es gibt drei verschiedene Arten von Metriken: Counter, Gauge und Histogram. Eine Counter Metrik zählt lediglich einen Wert hoch. Dies kann zum Beispiel ein Zähler für Anfragen oder Fehler sein. Gauge verhält sich wie ein Tachometer und zeigt stets den aktuellen Wert von zum Beispiel RAM oder CPU-Auslastung. Der Histogram Typ wird für Werte verwendet, die über einen Zeitraum akkumuliert werden. Das können beispielsweise Anfragen an einen Webservice oder die Durchschnittliche Antwortzeit sein (Prometheus 2021h).

Die so eingesammelten Metriken werden standardmäßig in einer lokalen TSDB, zu sehen mittig in Abbildung 2.14, in einem für Prometheus optimierten Format gespeichert. Es ist zwar möglich auch externe Dienste für das Speichern zu verwenden, jedoch wird empfohlen, die Daten hauptsächlich lokal vorzuhalten. Dadurch ist beispielsweise eine Auswertung von Alert Regeln, selbst bei einer nicht erreichbaren externen Datenbank, möglich. Die standardmäßige Zeit, für die Metriken gespeichert werden, beträgt 15 Tage. Danach werden die ältesten Metriken gelöscht, wobei sich dieser Zeitraum beliebig anpassen lässt. Hier muss beachtet werden, dass im Normalfall nur die Ressourcen eines einzigen Servers zur Verfügung stehen, daher können hier bei zu vielen Targets oder bei Ausfall von Speichermedien, Probleme entstehen (Prometheus 2021k).

TSDBs folgen einem gänzlich anderen Funktionsprinzip als herkömmliche relationale Datenbanksysteme. Daten werden in der Regel in relativ kurzen Abständen gespeichert, und anstatt bestehende Daten zu verändern, werden normalerweise neue Daten hinzugefügt. Dies liegt vor allem an der Natur der Daten, welche für gewöhnlich in TSDBs gespeichert werden, so bieten sich vor allem Daten an, welche in großer Menge und Frequenz anfallen. Ein Beispiel hierfür wären Sensordaten (Dix 2021). Die zu speichernden Daten werden, anders als in relationalen Datenbanken, mithilfe eines Timestamps und eventuell vergebenen Tags gespeichert. Im

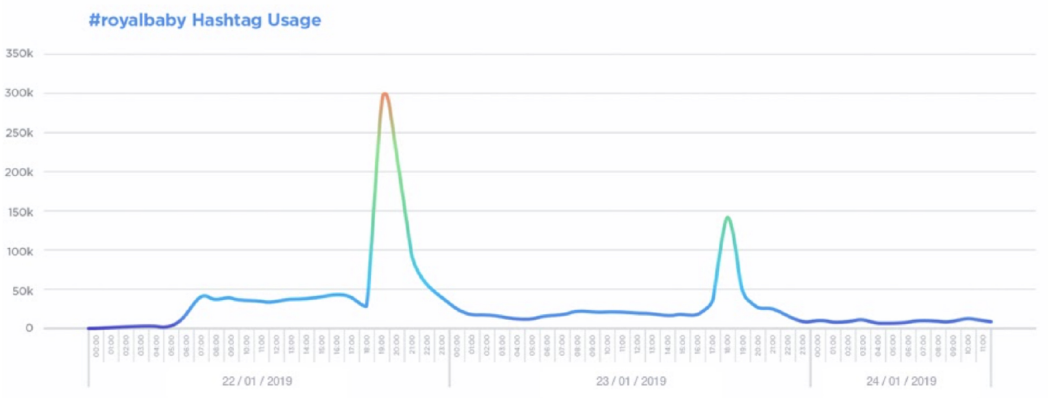

Abbildung 2.15: Beispiel für die Darstellung von Time Series Daten zur Häufigkeit von Twitter Hashtags über einen Zeitraum (Dix 2021).

Fall von Prometheus werden beispielsweise standardmäßig alle 15 Sekunden Daten von den jeweiligen Targets abgerufen und gespeichert (Prometheus 2021d). Diese Daten bilden über einen gewissen Zeitraum hinweg eine Zeitreihe oder auch Time Series und werden ebenfalls auf andere Art und Weise abgefragt, als es bei relationalen Datenbanken der Fall ist. Bei der Abfrage wird für gewöhnlich spezifiziert, welche Zeitreihe abgefragt werden soll, beispielsweise Sensor A. Zudem werden ein Zeitraum und eventuelle weitere Filter wie Tags definiert. Das Ergebnis einer solchen Abfrage liefert alle Datenpunkte mit den zugehörigen Timestamps für den angegebenen Zeitraum, wie am Beispiel in Abbildung 2.15 zu sehen (Dix 2021). Diese Daten können nun als Diagramme dargestellt werden. Hierfür kann beispielsweise Grafana verwendet werden (Unterabschnitt 2.7.4).

Neben dem Einsammeln der Metriken können auch sogenannte Recording- und Alerting Regeln definiert werden. Der Prometheus Server wertet in den Recording Regeln definierte Abfragen aus und speichert die Ergebnisse zwischen, um bei komplexen Abfragen die Antwortzeiten zu verbessern. Anhand der definierten Alerting Regeln wird dagegen beim Über- oder Unterschreiten von Grenzwerten ein Alert ausgelöst. Diese Alerts werden an die registrierten Alertmanager (Unterabschnitt 2.7.3), zu sehen rechts oben in Abbildung 2.14, gesendet, welche dann das Gruppieren und Versenden von Benachrichtigungen übernehmen (Prometheus 2021a).

Um Prometheus ohne zusätzliche Hilfsmittel in einem HA Setup zu betreiben, bleibt nur die Verwendung der Aktiv-Aktiv-Redundanz. So können mehrere Prometheus Instanzen die gleichen Targets abfragen und diese Daten in ihren lokalen TSDBs speichern (Prometheus 2021g). Eine horizontale Skalierung ist möglich,

indem die Prometheus Instanzen nur eine Untermenge der insgesamt verfügbaren Targets abfragen. Eine globale Sicht für die Metriken mehrerer Instanzen kann über die Federation-Funktion geschaffen werden. Über die */federate* Schnittstelle der untergeordneten Prometheus Instanzen kann eine globale Instanz Metriken mehrerer Cluster aggregieren (Prometheus 2021f).

Bei einem reinen Prometheus Setup muss bei der Skalierung oder dem Betrieb mehrerer Instanzen zum Erreichen von Ausfallsicherheit beachtet werden, dass die gesammelten Metriken nur lokal gespeichert werden und keinerlei Mechanismus zur Deduplizierung bei der Abfrage zur Verfügung steht. So ist der Ansatz einer globalen Sicht mithilfe der Federation-Funktion nur bis zu einem gewissen Grad möglich, da einerseits Metriken nicht dedupliziert werden und andererseits die globale Instanz durch die ihr zur Verfügung stehenden endlichen Ressourcen eingeschränkt ist (Prometheus 2021f).

2.7.2 Thanos

Thanos ist ein von Improbable gegründetes unter der Apache License 2.0 linzenziertes Open Source Projekt[17], an dem Entwickler von namhaften Firmen wie Red Hat oder Grafana Labs beteiligt sind. Das Ziel des Projekts ist es, die Werkzeuge zu liefern, um Prometheus mit überschaubarem Aufwand in einem HA Setup betreiben zu können und gleichzeitig ein zentrales Frontend für die Datenabfrage von mehreren Clustern zu bieten. Neben der zentralen Abfrage sind die weiteren Hauptmerkmale die theoretisch unlimitierte Aufbewahrung von Metriken mit Hilfe von externen Speicherlösungen wie S3, sowie das Komprimieren und Downsamplen der gesammelten Metriken (Lucas Serven 2019). Ein Beispiel für ein solches Setup liefert Abbildung 2.16.

Beim Downsampling werden einfach gesagt ältere Daten so neu gruppiert, dass die Präzision der Daten für kleine Zeiträume abnimmt, dafür aber der Trend über den größeren Zeitraum weiter erhalten bleibt. In der Praxis wird Downsampling oft verwendet, um die Abfrage von Daten für größere Zeiträume zu beschleunigen, da zum Beispiel bei einer Jahresansicht Daten im Sekundenbereich der einzelnen Tage oder Wochen irrelevant sind, aber die Abfrage wesentlich verlangsamen würden (Liu u. a. 2020).

[17] https://github.com/thanos-io/thanos

Alle Komponenten von Thanos sind grundsätzlich mit den APIs von Prometheus kompatibel, beziehungsweise implementieren eben diese APIs. So kann zum Beispiel in einer Grafana (Unterabschnitt 2.7.4) Instanz das Thanos Query Frontend als Prometheus Datenquelle angegeben und so Daten von mehreren Clustern sowie aus dem S3 Speicher abgerufen werden (Thanos 2021a).

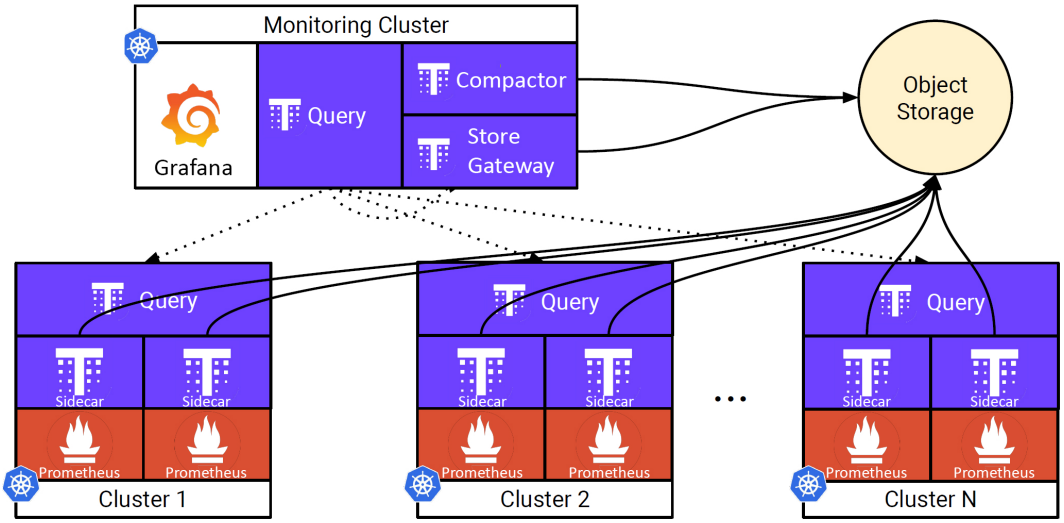

Abbildung 2.16: Exemplarisches über mehrere Cluster verteiltes High Level Prometheus Setup mit Thanos Sidecars (Lucas Serven 2019).

Den Keep it simple, stupid (KISS) und Unix Prinzipien folgend, übernimmt jede der Thanos Komponenten eine spezielle Aufgabe. Bisher existieren die folgenden Komponenten (Thanos 2021a):

- Sidecar: stellt eine Verbindung zu Prometheus her, liest dessen Daten zur Abfrage und/oder lädt sie in den Objektspeicher hoch.
- Store Gateway: dient der Bereitstellung von Metriken innerhalb eines Objektspeicher Buckets.
- Compactor: komprimiert, downsampled und wendet die Aufbewahrungsrichtlinien auf die im Objektspeicher Bucket gespeicherten Daten an. Sollte als Singleton Instanz pro Bucket betrieben werden.
- Receiver: empfängt Daten von Prometheus remote-write (Verwendung eines externen Time Series Speichers), stellt sie zur Verfügung und/oder lädt sie in den Objektspeicher hoch.
- Ruler/Rule: wertet Recording- und Alertregeln für Daten in Thanos für die Exposition und/oder den Upload aus.

- Querier/Query: implementiert die v1-API von Prometheus, um Daten aus den zugrunde liegenden Komponenten zu aggregieren.
- Query Frontend: implementiert die v1-API von Prometheus als Proxy für Query mit Zwischenspeicherung der Antwort und optionaler Aufteilung nach Abfragetagen.

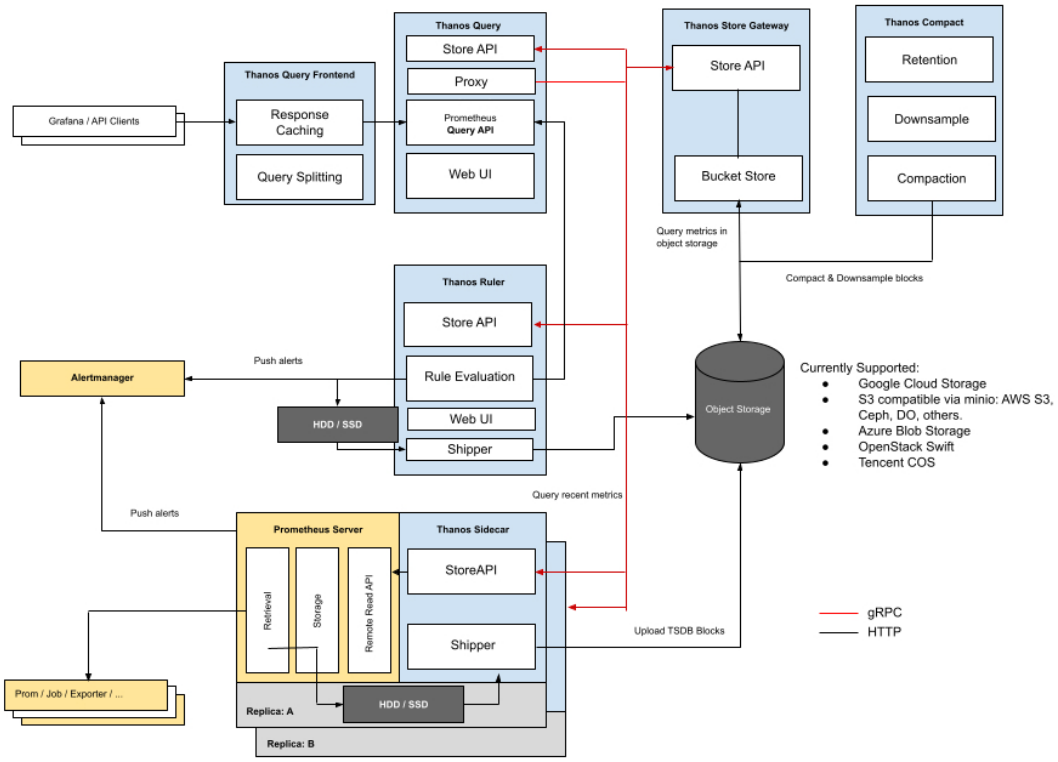

Abbildung 2.17: Architektur von Thanos in Verbindung mit Prometheus unter Verwendung von Thanos Sidecars (Thanos 2021a).

Abbildung 2.16 zeigt eine globale Sicht auf ein über mehrere Cluster verteiltes Prometheus Setup unter Verwendung von Thanos Sidecars, während Abbildung 2.17 und Abbildung 2.18 das Zusammenspiel der verschiedenen Komponenten verdeutlichen. Abbildung 2.17 zeigt ein Prometheus Setup in Verbindung mit allen Thanos Komponenten unter Verwendung von Thanos Sidecars. Eine Alternative hierzu wäre die Verwendung des Thanos Receivers an Stelle der Sidecars wie in Abbildung 2.18, um die eingesammelten Metriken direkt von den Prometheus Instanzen an den Thanos Receiver schicken zu lassen. Diese Methode birgt jedoch den Nachteil, dass die Netzwerkkommunikation zwischen den Prometheus Instanzen und dem Thanos Receiver immer vorhanden sein muss. Die Speicherung gesammelter Metriken wäre somit bei einem Netzwerkausfall nicht mehr möglich.

Die drei essenziellen Komponenten sind hier die Sidecars beziehungsweise Receiver sowie der Querier und das Store Gateway.

Die Sidecars stellen die Thanos Store API für den Querier zur Verfügung und laden gleichzeitig die von Prometheus gesammelten Metriken in den Objektspeicher hoch. Der Querier verfügt über eine Weboberfläche (Abbildung 2.19) zum Absetzen von PromQL Abfragen sowie eine Store API, somit können Querier beliebig in der Abfragehierarchie angeordnet werden (Thanos 2021a).

Das Store Gateway bietet ebenfalls eine Weboberfläche an, hier sind jedoch nur die aktuell im Objektspeicher verfügbaren Blöcke zu sehen. Die Hauptfunktion besteht darin, die Daten im Objektspeicher über die Store API verfügbar zu machen. Die Kommunikation unter den Thanos Komponenten erfolgt vorrangig über Google Remote Procedure Calls (gRPC) (Thanos 2021a).

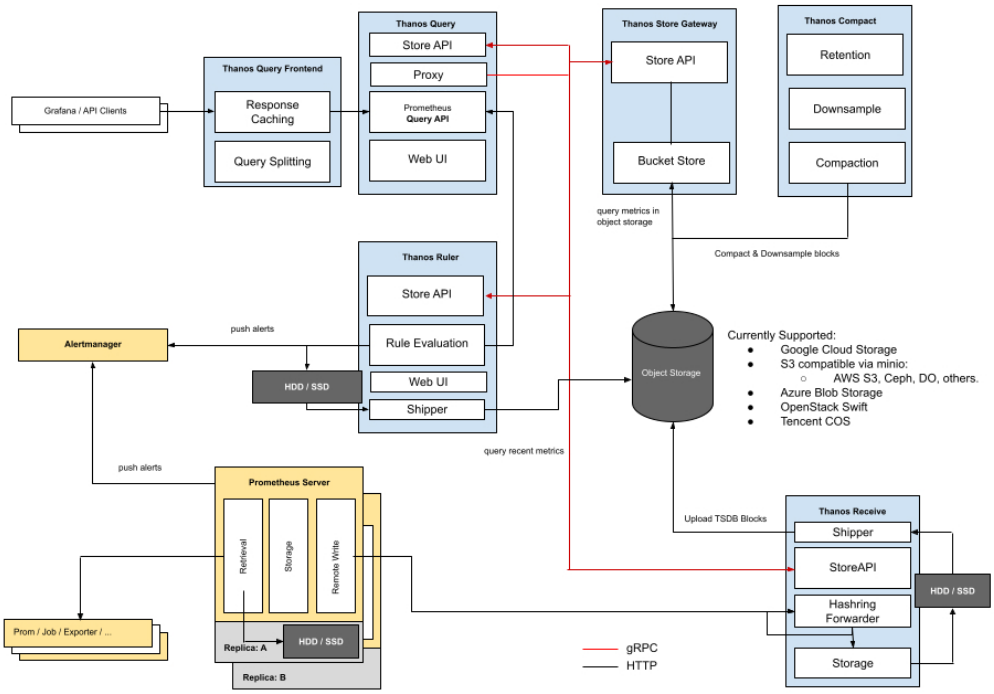

Abbildung 2.18: Architektur von Thanos in Verbindung mit Prometheus unter Verwendung des Thanos Receivers (Thanos 2021a).

Je nach Ausprägung und Größe des Setups kann es sinnvoll sein, nur einzelne Komponenten einzusetzen. Bei über mehrere Regionen und Rechenzentren verteilten Setups bringt es so durchaus Vorteile, das Query Frontend zum Zwischenspeichern von Ergebnissen einzusetzen und den Compactor zu verwenden, um die histori-

schen Daten im Objektspeicher zu downsamplen. So können Abfragen von großen Zeiträumen, oder Daten, welche nicht mehr in den lokalen TSDBs der Prometheus Instanzen vorhanden sind, beschleunigt werden. In kleineren Setups mit kürzeren Aufbewahrungszeiten der Metriken hingegen, bringt der Compactor nur einen geringen Vorteil, da die betrachteten Zeiträume und damit verbundenen Datenmengen eher klein sind. Der Ruler bezieht hier eine besondere Stellung, denn er bietet zwar eine Weboberfläche und die Store API an, ist aber primär dafür vorgesehen, globale Recording- und Alerting Regeln auszuwerten (Thanos 2021a).

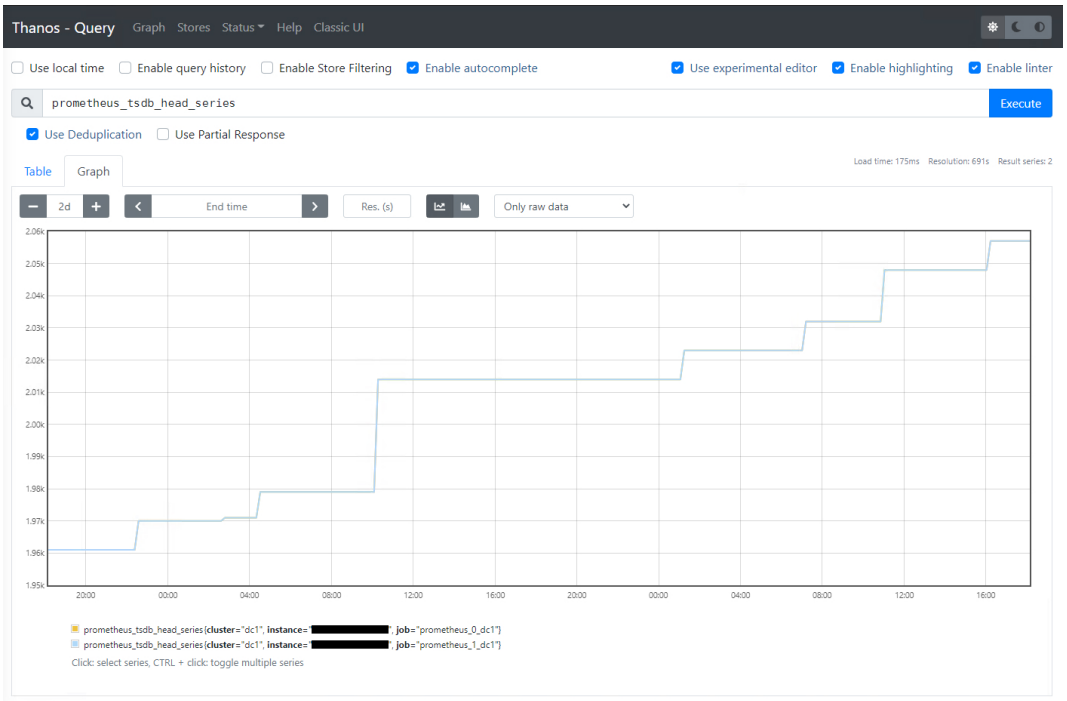

Abbildung 2.19: Screenshot der Thanos Querier Weboberfläche.

Grundsätzlich ist es auch möglich, nur einzelne Komponenten zu betreiben, oder aber ein bestehendes Prometheus Setup nachträglich mit Thanos zu erweitern, da die einzige direkte Verbindung von Thanos zu Prometheus durch die jeweiligen Sidecars beziehungsweise den Receiver stattfindet. Die weitere Kommunikation unter den Thanos Komponenten läuft vorrangig über gRPC ab (Thanos 2021a).

Mithilfe dieser Komponenten wird es relativ einfach, Prometheus in einem HA Setup zu betreiben. So können beliebig viele Prometheus Instanzen pro Cluster betrieben werden, vorzugsweise mit Aktiv-Aktiv-Redundanz. Diese Prometheus Instanzen fragen alle die gleichen Targets ab, was zunächst einmal duplizierte Daten erzeugt. Hier kommen die Thanos Querier ins Spiel. Diese können, anhand der von den Prometheus Instanzen für ihre Metriken gesetzten Labels, die doppelten Da-

ten für die Ergebnisse der Abfragen deduplizieren. Damit dies funktioniert, muss für jede Querier Instanz eingestellt werden, anhand welches Labels die deduplizierung stattfinden soll. Üblicherweise bietet sich hierfür das *replica* Label an (Thanos 2021a).

2.7.3 Alertmanager

Der Alertmanager ist ein Teil des Prometheus Projekts[18] und kann wie Prometheus auch entweder als einzelne Instanz oder, wie in Abbildung 2.20 zu sehen, in einem HA Setup betrieben werden. Anders als Prometheus lässt sich der Alertmanager jedoch leichter in einem solchen Setup betreiben, da hier bei mehreren Instanzen lediglich ein paar Startparameter notwendig sind, um den jeweiligen Instanzen die Adressen der anderen Instanzen mitzuteilen. Die Alertmanager bilden dann mit ihren jeweiligen Peers und mithilfe eines Gossip Protokolls, wie in Unterabschnitt 2.4.3 beschrieben, ein Cluster. Eine Lastverteilung, für die von den Prometheus Instanzen verschickten Alerts, sollte hier nicht stattfinden, da dies durch das Alertmanager Cluster geregelt wird (Prometheus 2021b).

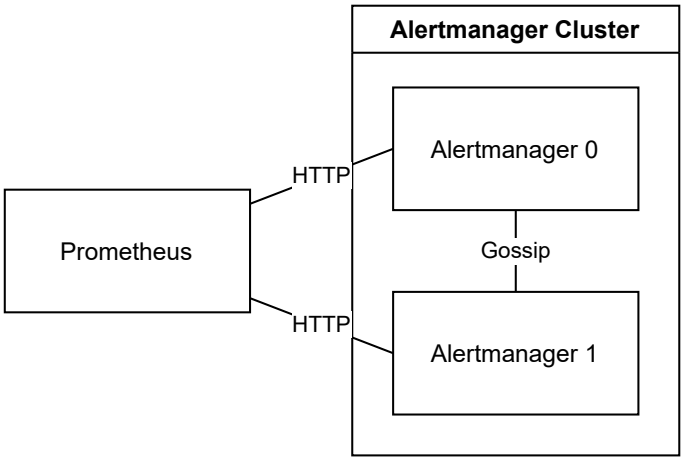

Abbildung 2.20: Diagramm zur Veranschaulichung der Architektur eines Alertmanager Clusters.

Da Prometheus selbst keine Funktion bietet, um Benachrichtigungen zu versenden, senden die Prometheus Instanzen ihre Alerts an ihre registrierten Alertmanager. In einem HA Setup ist es hier außerdem wichtig, alle Alertmanager Instanzen bei den Prometheus Instanzen zu registrieren. Die Deduplizierung und Gruppierung von Benachrichtigungen übernehmen die Alertmanager. Im Alertmanager selbst werden

[18] https://github.com/prometheus/alertmanager

die jeweiligen Integrationen zum Versenden von Benachrichtigungen konfiguriert. Das können beispielsweise E-Mails oder auch Nachrichten über Dienste wie Slack oder Microsoft Teams sein (Prometheus 2021b).

Neben dem Gruppieren von Nachrichten, um zum Beispiel beim Ausfall eines ganzen Clusters nicht hunderte einzelne Benachrichtigungen zu bekommen, ist es auch möglich für eben so einen Fall sogenannte Inhibition Rules zu definieren. Mit Hilfe dieser Regeln kann der Alertmanager beim Eintreffen bestimmter Alerts, zum Beispiel einem Alert, dass Cluster A nicht mehr erreichbar ist, alle weiteren Benachrichtigungen, welche mit dieser in Zusammenhang stehen, stumm schalten. Die Einstellungen für Gruppierung und Inhibition werden in der Konfigurationsdatei der Alertmanager festgelegt. Für den Fall, dass eine oder mehrere spezielle Benachrichtigungen für einen Zeitraum stumm geschaltet werden sollen, gibt es auch die Möglichkeit, über die Weboberfläche der Alertmanager sogenannte Silences zu definieren. Diese werden ebenfalls unter den Instanzen synchronisiert (Prometheus 2021b).

2.7.4 Grafana

Grafana ist ein Open Source Projekt[19] welches von Grafana Labs unter der GNU Affero General Public License v3.0 entwickelt wird. Neben der Möglichkeit, eine eigene Instanz in der Open Source oder der kostenpflichtigen Enterprise Variante zu betreiben, kann auch eine Cloud Instanz direkt bei Grafana Labs gemietet werden. Die Hauptfunktionalität von Grafana besteht darin, Daten aus verschiedenen Datenquellen in Dashboards darzustellen. Eine solche Datenquelle kann zum Beispiel eine Prometheus Instanz oder eine Datenbank sein (Grafana 2021c).

Außerdem bietet Grafana auch die Möglichkeit, von der Community erstellte Dashboards zu importieren oder selbst erstellte Dashboards zu teilen. Zusätzlich lassen sich auch Alerts über mehrere Datenquellen hinweg über die Weboberfläche einrichten oder Daten in Echtzeit über Websockets streamen (Grafana 2021c).

Grafanas Weboberfläche geht über die von Prometheus und Thanos gebotenen Möglichkeiten weit hinaus. Prometheus und Thanos ermöglichen nur die Abfrage der Rohdaten und deren Visualisierung in einfachsten Diagrammen, wie bereits in Abbildung 2.19 dargestellt. Grafana hingegen ermöglicht die Gestaltung komplexer

[19] https://github.com/grafana/grafana

Abbildung 2.21: Screenshot der Grafana Weboberfläche auf https://play.grafana.org.

individueller Dashboards, siehe Abbildung 2.21. Darüber hinaus erlaubt es eine Benutzer- und Zugriffsverwaltung sowie die Anbindung an verschiedene Verzeichnisdienste wie etwa Lightweight Directory Access Protocol (LDAP) (Grafana 2021b; Grafana 2021a).

2.7.5 Organizr

Die Hauptfunktionalität von Organizr besteht darin, verschiedene Webseiten über iFrames einzubinden und über Tabs in der Organizr Weboberfläche aufrufbar zu machen, ohne einen neuen Tab im Webbrowser öffnen zu müssen, wie in Abbildung 2.22 zu sehen. Außerdem ist für die Benutzerverwaltung eine LDAP oder Active Directory Anbindung möglich. So können Rechtestrukturen wie Gruppenzugehörigkeiten, aus bestehenden Verzeichnisdiensten abgefragt werden, um zum Beispiel bestimmte Tabs nur bestimmten Gruppen zugänglich zu machen (HaliaNElf 2019).

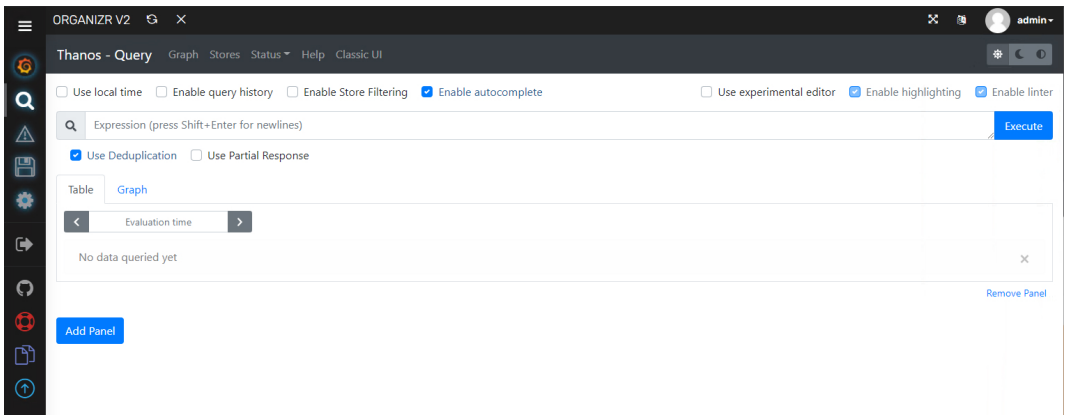

Abbildung 2.22: Screenshot der Organizr Weboberfläche, zu sehen ist der Thanos Querier Tab.

Kapitel 3

Vorgehensweise

In diesem Kapitel wird die Vorgehensweise für die Erarbeitung der Inhalte der kommenden Kapitel beschrieben. Dazu gehört insbesondere die Methodik zur Ermittlung der Anforderungen an die Monitoring Lösung, da diese Anforderungen die Basis für die weitere Konzeption bilden.

3.1 Anforderungsanalyse

Im Zuge der Anforderungsanalyse in Kapitel 4 wird ein qualitatives Experteninterview mit den Mitgliedern des Operations Teams durchgeführt, welche gleichzeitig die Rolle der Stakeholder besetzen. Das Interview und die Fragen sind offen gestaltet, um möglichst viele Ideen und Vorstellungen der Befragten in Erfahrung zu bringen. Folgende Fragen werden im Interview gestellt oder als Leitfaden verwendet:

1. Was sind Ihre täglichen Aufgaben, besonders im Bereich des Kontexts dieses Interviews, und welche Tools verwenden Sie aktuell bzw. wie gehen Sie z.B. bei einem Systemausfall vor?
2. Welche Probleme existieren mit momentan verwendeten Lösungen, und weshalb wurde entschieden, die nun zu konzipierende Lösung umzusetzen?
3. Was denken Sie, wie das System ihre Arbeit erleichtern wird, bzw. was erwarten/erhoffen Sie sich?
4. Was verstehen Sie unter dem Aspekt Sicherheit im Kontext des Systems?

5. Was verstehen Sie unter dem Aspekt High Availability im Kontext des Systems?
6. Welche Informationen hoffen Sie durch das System zu gewinnen bzw. zu erfassen?
7. Wie stellen sie sich die alltägliche Verwendung des Systems vor?
8. Da bei dem umzusetzenden System kontinuierlich Daten in Form von Metriken erhoben werden, welche teils lokal gehalten aber auch in einen Objektspeicher ausgelagert werden, stellt sich die Frage, für welchen Zeitraum Sie gerne unabhängig vom externen Anbieter sein möchten. Also welchen Zeitraum sie im eigenen Rechenzentrum vorhalten möchten?
9. Auf welche Art und Weise würden Sie das System gerne betreiben, das heißt welche Optionen kommen für Sie für das Hosting in Frage?
10. Wie könnte/sollte sich das System Ihrer Meinung nach in Zukunft entwickeln?

Die Fragen eins bis drei bilden hier den Einstieg in das Interview und sollen dazu dienen, die jetzige Situation einzuschätzen und die aktuellen Probleme oder Unzulänglichkeiten der bisherigen Lösungen in Erfahrung zu bringen. Die Fragen vier bis neun stellen den Hauptteil des Interviews dar. Hier geht es darum, möglichst viele der Wünsche und Vorstellungen der Stakeholder in Erfahrung zu bringen, um so Anwendungsfälle, funktionale Anforderungen sowie Randbedingungen und Qualitätsanforderungen ableiten und priorisieren zu können. Die letzte Frage dient dazu eine Idee davon zu bekommen, wie sich die Lösung in Zukunft weiterentwickeln könnte, beziehungsweise festzustellen was die Vision der Stakeholder ist.

Die Interviews werden über das firmenintern verwendete Microsoft Teams durchgeführt und das Gespräch aufgezeichnet. Anschließend wird die Aufnahme für die firmeninterne Verwendung auf Microsofts Stream Plattform hochgeladen, um die Transkripte automatisch zu erzeugen. Diese Transkripte werden danach von Hand nachbearbeitet, um Ungenauigkeiten und Fehler bei der Transkription auszubessern.

Für die Ermittlung der Anforderungen und Anwendungsfälle werden beispielsweise die Satzschablone für Anforderungen und andere Herangehensweisen verwendet, wie sie in Ebert 2019 beschrieben werden. Für die Priorisierung der Anforderungen werden die Abstufungen Gering, Mittel und Hoch verwendet.

3.2 Systemarchitektur

Im Rahmen von Kapitel 5 wird zunächst der Ist-Zustand der Infrastruktur erfasst beziehungsweise analysiert. Hierfür wird die Infrastruktur mithilfe eines Verteilungsdiagramms dargestellt und die einzelnen Komponenten und deren Funktion erklärt. Außerdem werden die für den Betrieb der Infrastruktur essenziellen Dienste vorgestellt.

Im zweiten Abschnitt werden die Integrations- und Betriebsmöglichkeiten für die Monitoring Lösung auf Grundlage der Experteninterviews und der bestehenden Infrastruktur analysiert.

Im dritten Abschnitt wird der Systementwurf für die Monitoring Lösung anhand der zuvor ermittelten Informationen erarbeitet. Unter anderem wird das Zusammenspiel der verschiedenen Komponenten innerhalb der verschiedenen Umgebungen mithilfe eines Verteilungsdiagramms dargestellt sowie auf Details wie das Konfigurationsmanagement und dessen Umsetzung eingegangen.

3.3 Machbarkeitsnachweis

In Kapitel 6 wird zunächst der Systementwurf unter Verwendung der ermittelten Betriebs- und Integrationsmöglichkeiten umgesetzt. Hier wird außerdem erklärt, welche Technologien zum Einsatz kommen und wie der Betrieb beziehungsweise das Konfigurationsmanagement in der Praxis funktioniert. Zusätzlich werden erste Systeme an das Monitoring angebunden.

Anschließend werden grundlegende Alerting Regeln definiert beziehungsweise vorgestellt und implementiert. Abschließend erfolgt die Konfiguration einiger grundlegender Dashboards innerhalb der Visualisierungslösung.

3.4 Evaluation der Umsetzung

In Kapitel 7 wird die Umsetzung des Systementwurfs anhand der Anforderungen aus Kapitel 4 evaluiert. Es wird anhand der Abnahmekriterien ermittelt, welche Anforderungen umgesetzt wurden und welche nicht.

Hierfür werden die Anforderungen in Tabellen dargestellt, um einen Überblick über die erfüllten und nicht erfüllten Anforderungen zu schaffen. Anschließend wird erklärt, wie und wieso die Anforderungen erfüllt beziehungsweise nicht erfüllt werden konnten.

Kapitel 4

Anforderungsanalyse

4.1 Anwendungsfälle

4.1.1 Auf Benutzeroberfläche zugreifen, um Daten abzufragen

Mitarbeiter greifen auf die Benutzeroberfläche der Monitoring Lösung zu, um sich mithilfe der Dashboards einen Überblick über die überwachten Systeme zu verschaffen.

Daraufhin werden die Anmeldedaten mit dem zentralen Verzeichnisdienst abgeglichen, und die Benutzeroberfläche stellt die für den Mitarbeiter aufgrund seiner Berechtigungen zugänglichen Daten dar.

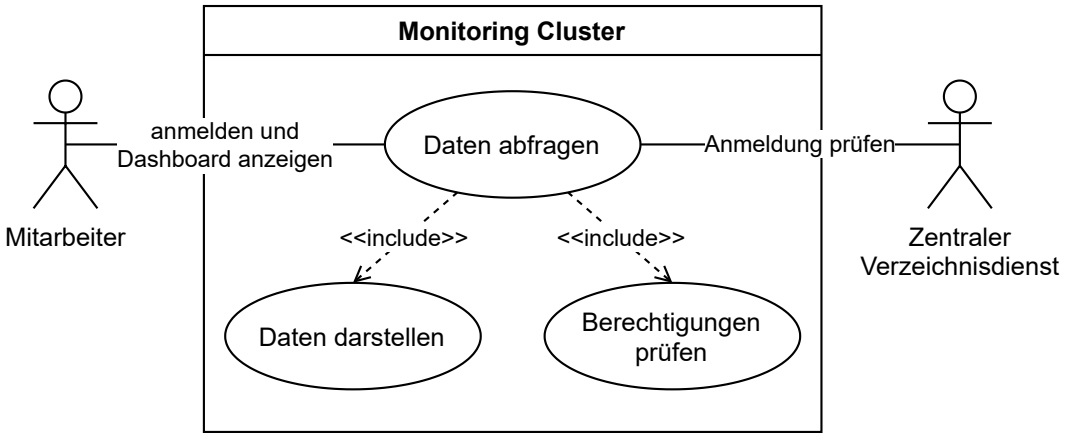

Abbildung 4.1: Anwendungsfalldiagramm zur Abfrage von Daten vom Monitoring System.

4.1.2 Benachrichtigungen erhalten

Mitarbeiter erhalten Benachrichtigungen anhand ihrer Projekt- beziehungsweise Gruppenzugehörigkeiten. Diese Benachrichtigungen werden über E-Mail und Microsoft Teams versandt.

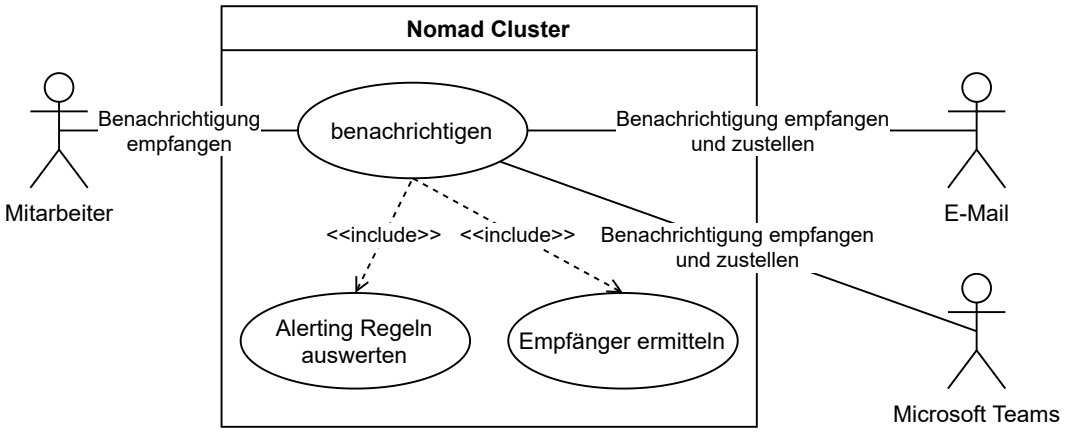

Abbildung 4.2: Anwendungsfalldiagramm zur Benachrichtigung eines Mitarbeiters bei Alerts.

4.1.3 Konfiguration anpassen

Ein Administrator ändert die Konfiguration einer oder mehrerer Monitoring Komponenten im Nomad Cluster und startet den CI/CD Job zur Ausführung des Nomad Jobs. Der Nomad Job wird ausgeführt und die Änderung der Konfiguration von Nomad erkannt. Daraufhin signalisiert Nomad dem jeweiligen Container die Änderung der Konfiguration, worauf dieser die Konfigurationsdatei neu einliest.

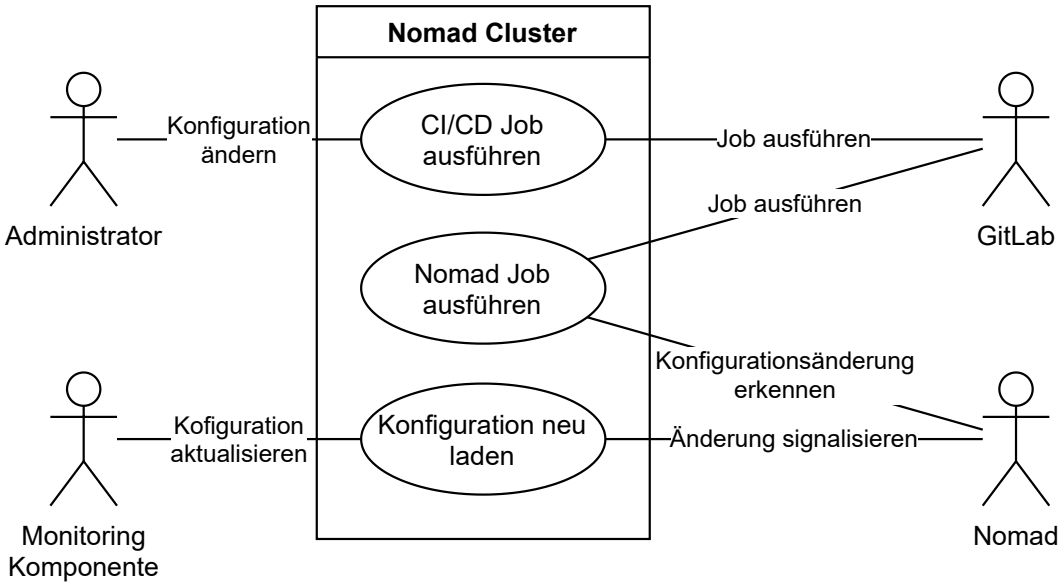

Abbildung 4.3: Anwendungsfalldiagramm für das Konfigurationsmanagement.

4.2 Funktionale Anforderungen

4.2.1 Monitoring von Hosts und Containern

Da die bestehende Infrastruktur sehr heterogen ist und klassische VMs sowie Container betrieben werden, ist es erforderlich, dass die Monitoring Lösung eine Möglichkeit bietet diese Systeme und Dienste zu überwachen. Aufgrund dessen wird diese Anforderung mit hoher Priorität eingestuft (Experte A 2021; Experte B 2021).

ID	FA1	**Prio**	Hoch
Titel	Monitoring von Hosts und Containern		
Herkunft	Experteninterviews		
Beschreibung			
Die Monitoring Lösung soll so flexibel sein, dass ein Monitoring von Containern wie auch Hosts gleichermaßen möglich ist.			
Abnahmekriterium			
Die Monitoring Lösung bietet die Möglichkeit Metriken von Hosts und Containern einzusammeln.			

Tabelle 4.1: FA1 – Monitoring von Hosts und Containern

4.2.2 Monitoring von Kundensystemen

Langfristig sollen auch Kundensysteme an das Monitoring angebunden werden, um den Kunden bessere Informationen zur Nutzung ihrer Systeme und Dienste bieten zu können und so einen Wettbewerbsvorteil zu schaffen. Da dies momentan noch nicht in Planung ist, wird diese Anforderung mit mittlerer Priorität eingestuft (Experte A 2021; Experte B 2021).

ID	FA2	**Prio**	Mittel
Titel	Monitoring von Kundensystemen		
Herkunft	Experteninterviews		
Beschreibung			
Die Systemarchitektur der Monitoring Lösung soll es ermöglichen, auch externe Systeme zu überwachen.			
Abnahmekriterium			
Die Systemarchitektur der Monitoring Lösung erlaubt es, auch externe Systeme anzubinden, beziehungsweise Metriken von diesen zu sammeln.			

Tabelle 4.2: FA2 – Monitoring von Kundensystemen

4.2.3 Monitoring von Systemen vor Ort

Da die Infrastruktur nicht nur in der Cloud betrieben wird und wie in Unterabschnitt 4.2.1 beschrieben daher sehr heterogen ist, muss die Monitoring Lösung auch Möglichkeiten bieten, um Systeme im Firmensitz zu überwachen. Hierzu zählen unter anderem lokal betriebene Server sowie Router oder Switches. Da diese Komponenten ebenfalls ein wichtiger Bestandteil der Infrastruktur sind, wird diese Anforderung mit hoher Priorität eingestuft (Experte A 2021; Experte B 2021).

ID	FA3	**Prio**	Hoch
Titel	Monitoring von Systemen vor Ort		
Herkunft	Experteninterviews		
Beschreibung			
Die Architektur der Monitoring Lösung muss es erlauben, auch Systeme vor Ort zu überwachen, welche nicht in der Cloud betrieben werden.			
Abnahmekriterium			
Systeme vor Ort können aufgrund der Systemarchitektur ebenfalls überwacht werden.			

Tabelle 4.3: FA3 – Monitoring von Systemen vor Ort

4.2.4 Authentifizierung für die Benutzeroberflächen

Weil die verschiedenen Monitoring Komponenten keine Authentifizierung beziehungsweise nur eine experimentelle Authentifizierung der Benutzeroberflächen erlauben, ist es von größter Wichtigkeit diese abzusichern. Daher wird diese Anforderung mit hoher Priorität eingestuft. Für diese Absicherung ist gegebenenfalls auch eine Virtual Private Network (VPN) Verbindung zu nutzen, da so der Zugriff von dritten ausgeschlossen werden kann (Experte A 2021; Experte B 2021).

ID	FA4	**Prio**	Hoch
Titel	Authentifizierung für die Benutzeroberflächen		
Herkunft	Experteninterviews		
Beschreibung			
Die Benutzeroberflächen der Monitoring Lösung müssen durch Authentifizierung geschützt sein.			
Abnahmekriterium			
Die Benutzeroberflächen der Monitoring Lösung sind nur nach Authentifizierung zugänglich.			

Tabelle 4.4: FA4 – Authentifizierung für die Benutzeroberflächen

4.2.5 Abbildung bestehender Rechtestrukturen

Da mittelfristig auch neu entwickelte Anwendungen mit an das Monitoring angebunden werden sollen, um den Projektteams einen besseren Überblick über die Systeme zu vermitteln und bei Problemen eine schnelle Reaktion zu ermöglichen, sollen bestehende Rechtestrukturen im LDAP beziehungsweise dem Active Directory abgebildet werden können. Diese Rechtestrukturen sollen für die Benutzeroberflächen sowie die Benachrichtigungen abbildbar sein. Da eine gesonderte beziehungsweise doppelte Nutzerverwaltung hier nicht gewünscht ist, ist diese Anforderung mit hoher Priorität eingestuft (Experte A 2021; Experte B 2021).

ID	FA5	**Prio**	Hoch
Titel	Abbildung bestehender Rechtestrukturen		
Herkunft	Experteninterviews		

Beschreibung
Um auch Projektteams den Zugang zu ermöglichen, beziehungsweise ihnen relevante Benachrichtigungen zukommen lassen zu können, sollen sich bestehende Rechtestrukturen abbilden lassen. Zum Beispiel mithilfe einer Active Directory oder LDAP Anbindung.

Abnahmekriterium
Rechtestrukturen lassen sich für den Zugriff auf die Benutzeroberflächen sowie für Benachrichtigungen abbilden.

Tabelle 4.5: FA5 – Abbildung bestehender Rechtestrukturen

4.2.6 Benachrichtigungen über Microsoft Teams

Für das Empfangen von Benachrichtigungen soll primär das firmenintern verwendete Microsoft Teams verwendet werden. Da Microsoft Teams intern das primäre Kommunikationswerkzeug darstellt und jeder Mitarbeiter so erreicht werden kann, ist diese Anforderung mit hoher Priorität eingestuft (Experte A 2021; Experte B 2021).

ID	FA6	**Prio**	Hoch
Titel	Benachrichtigungen über Microsoft Teams		
Herkunft	Experteninterviews		

Beschreibung
Die Alerting Komponente der Monitoring Lösung soll Benachrichtigungen über Microsoft Teams versenden können.

Abnahmekriterium
Benachrichtigungen können über Microsoft Teams versendet werden.

Tabelle 4.6: FA6 – Benachrichtigungen über Microsoft Teams

4.2.7 Benachrichtigungen über E-Mail

Als sekundärer Benachrichtigungskanal zu Microsoft Teams (Unterabschnitt 4.2.6) sollen ebenfalls E-Mails versendet werden. Dies dient in erster Linie der Absicherung und ist daher mit mittlerer Priorität eingestuft (Experte A 2021; Experte B 2021).

ID FA7	**Prio** Mittel
Titel	Benachrichtigungen über E-Mail
Herkunft	Experteninterviews
Beschreibung	Die Alerting Komponente der Monitoring Lösung soll Benachrichtigungen über E-Mail versenden können.
Abnahmekriterium	Benachrichtigungen können über E-Mail versandt werden.

Tabelle 4.7: FA7 – Benachrichtigungen über E-Mail

4.2.8 Erstellung verschiedener Dashboards

Die Visualisierungslösung soll das Erstellen verschiedener Dashboards erlauben. Dies ist insbesondere deshalb wichtig, weil die verschiedenen Projektteams, wie in Unterabschnitt 4.2.5 beschrieben, nur Zugang zu den jeweils für sie relevanten Dashboards erhalten sollen. Außerdem sollen Dashboards für verschiedene Bereiche, wie zum Beispiel Dienste oder Infrastruktur, erstellt werden können. Da hier potenziell auch Dashboards mit Daten zu Kundensystemen erstellt werden, ist es besonders wichtig, diese von anderen Dashboards trennen zu können. Daher wird diese Anforderung mit hoher Priorität eingestuft (Experte A 2021; Experte B 2021).

ID FA8	**Prio** Hoch
Titel	Erstellung verschiedener Dashboards
Herkunft	Experteninterviews
Beschreibung	Die Darstellende Komponente der Monitoring Lösung soll die Erstellung verschiedener Dashboards erlauben.
Abnahmekriterium	Es ist möglich, in der Visualisierungslösung entweder über die Weboberfläche oder über die Konfiguration verschiedene Dashboards anzulegen.

Tabelle 4.8: FA8 – Erstellung verschiedener Dashboards

4.2.9 Gruppierungen innerhalb der Dashboards

Aufgrund der Anzahl der Systeme, welche bereits existieren, soll eine Gruppierung innerhalb der Dashboards möglich sein, um einen besseren Überblick wahren zu

können. Da dies vor allem für die Überwachung interner Systeme wichtig ist, wird diese Anforderung mit hoher Priorität eingestuft (Experte A 2021).

ID FA9	**Prio** Hoch
Titel	Gruppierungen innerhalb der Dashboards
Herkunft	Experteninterviews
Beschreibung	Die Darstellende Komponente der Monitoring Lösung soll das Anlegen von Gruppierungen unterstützen.
Abnahmekriterium	Es ist möglich, Elemente innerhalb eines Dashboards zu gruppieren.

Tabelle 4.9: FA9 – Gruppierungen innerhalb der Dashboards

4.2.10 „Ampel“ als Indikator für Systemzustand

Sobald Grenzwerte für die überwachten Systeme über- oder unterschritten werden, soll ein visueller Indikator dies anzeigen, sofern möglich. Dies ist wichtig, um eventuelle Probleme mit einem Blick erkennen zu können. Da bei Ausfällen ebenfalls Benachrichtigungen versendet werden sollen wie in Unterabschnitt 4.2.6 und Unterabschnitt 4.2.7 beschrieben, ist diese Anforderung lediglich eine Unterstützung und daher mit mittlerer Priorität eingestuft (Experte A 2021).

ID FA10	**Prio** Mittel
Titel	„Ampel“ als Indikator für Systemzustand
Herkunft	Experteninterviews
Beschreibung	Die Darstellende Komponente der Monitoring Lösung soll das Einstellen eines Indikators für den Zustand der überwachten Systeme in Form von visueller Kennzeichnung ermöglichen.
Abnahmekriterium	Es ist möglich, einen Indikator für den Gesamtzustand der Systeme einzustellen, beispielsweise realisiert als Ampel.

Tabelle 4.10: FA10 – „Ampel“ als Indikator für Systemzustand

4.2.11 Flexible Aufbewahrungsrichtlinien

Die historischen Metriken sollen nicht unbegrenzt gespeichert werden, da dies zumindest momentan keinen Mehrwert liefert (Unterabschnitt 4.3.6). Da sich dies in Zukunft jedoch ändern kann, müssen die Aufbewahrungsrichtlinien flexibel einstellbar sein. Daher ist diese Anforderung mit mittlerer Priorität eingestuft (Experte A 2021; Experte B 2021).

ID	FA11	**Prio**	Mittel
Titel	Flexible Aufbewahrungsrichtlinien		
Herkunft	Experteninterviews		
Beschreibung			
Die Aufbewahrungsrichtlinien für die Daten der Monitoring Lösung sollen flexibel einstellbar sein.			
Abnahmekriterium			
Die Aufbewahrungsrichtlinien lassen sich für lokale sowie Daten im S3 Speicher einstellen.			

Tabelle 4.11: FA11 – Flexible Aufbewahrungsrichtlinien

4.3 Randbedingungen

4.3.1 Hosting Anbieter

Die Monitoring Lösung soll bei IONOS betrieben werden. Andere Hosting-Anbieter sind nur in Ausnahmefällen zu verwenden. Dies ist darin begründet, dass bereits eine langjährige Geschäftsbeziehung mit IONOS besteht.

4.3.2 Technologie zum Einsammeln der Metriken

Für das Einsammeln der Metriken soll Prometheus verwendet werden, da Prometheus hierfür im Cloud Computing Umfeld den inoffiziellen Standard darstellt (CNCF 2020). Außerdem ist dies ebenfalls das Ergebnis der Experteninterviews, in denen beide Experten klargemacht haben, dass sie diesen Standard verwenden möchten, da sich dieser außerdem gut in die bestehende Infrastruktur integrieren lässt (Experte A 2021; Experte B 2021).

4.3.3 Technologie zur Skalierung der Monitoring Lösung

Da die Monitoring Lösung sich in Zukunft noch weiterentwickeln und gegebenenfalls mehrere Rechenzentren und Kundensysteme angebunden werden sollen, muss sichergestellt werden, dass die bekannten Probleme bei einem groß angelegten Prometheus Setup vermieden werden (Lucas Serven 2019; Lefevre 2021). Daher soll Thanos verwendet werden, um die Skalierung von Prometheus zu vereinfachen und eine globale Sicht zur Abfrage von Metriken zu schaffen.

4.3.4 Speicherlösung für historische Metriken

Zum Speichern der historischen Metriken soll ein S3 Objektspeicher verwendet werden, um die Ressourcen im eigenen virtuellen Rechenzentrum zu schonen und so unnötige Kosten und Aufwand zu vermeiden. Dies macht deshalb Sinn, weil somit IONOS für Backups und Verfügbarkeit der Daten verantwortlich ist.

4.3.5 Anbieter der Speicherlösung für historische Metriken

Als Speicherlösung soll das S3 Angebot von IONOS verwendet werden, da der Dienst bereits für andere Anwendungen verwendet wird und aufgrund der Experteninterviews als gesetzt zu betrachten ist (Experte A 2021; Experte B 2021).

4.3.6 Aufbewahrungszeiten der historischen Metriken

Metriken sollen für einen Zeitraum von 30 Tagen gespeichert werden, da längere Zeiträume für die momentan ausschließlich interne Verwendung nicht relevant sind (Experte A 2021; Experte B 2021).

4.4 Qualitätsanforderungen

4.4.1 Verteiltes System

Um Ausfallsicherheit und auch das lückenlose Sammeln von Metriken wie in Unterabschnitt 4.4.7 beschrieben zu erreichen, soll die Monitoring Lösung als verteiltes

System betrieben werden. Da dies ein zentraler Punkt ist wird diese Anforderung mit hoher Priorität eingestuft (Experte A 2021).

ID	QA1	**Prio**	Hoch
Titel	Verteiltes System		
Herkunft	Experteninterviews		

Beschreibung
Die Monitoring Lösung soll als verteiltes System betrieben werden.

Begründung
Um zum Beispiel nicht von einzelnen VMs abhängig zu sein und um eine gewisse Ausfallsicherheit und Skalierung zu ermöglichen, soll die Monitoring Lösung als verteiltes System betrieben werden.

Abnahmekriterium
Die Monitoring Lösung wird als verteiltes System betrieben.

Tabelle 4.12: QA1 – Verteiltes System

4.4.2 Skalierbarkeit des Systems

Um Anforderungen wie das Anbinden von Kundensystemen (Unterabschnitt 4.2.2) umsetzen und auch bei Vergrößerung der Infrastruktur noch eine problemlose Überwachung gewährleisten zu können, muss das System skalierbar sein. Daher ist diese Anforderung mit hoher Priorität eingestuft (Experte A 2021; Experte B 2021).

ID	QA2	**Prio**	Hoch
Titel	Skalierbarkeit des Systems		
Herkunft	Experteninterviews		

Beschreibung
Das System soll skalierbar sein.

Begründung
Um auch bei zunehmender Anzahl der Systeme noch ein lückenloses Monitoring zu ermöglichen, soll das System skalierbar sein.

Abnahmekriterium
Das System lässt sich beliebig skalieren, um den Anforderungen gerecht zu werden.

Tabelle 4.13: QA2 – Skalierbarkeit des Systems

4.4.3 Ein Prometheus HA Paar pro Cluster oder Einheit

Mit Blick auf Unterabschnitt 4.4.7 und Unterabschnitt 4.4.4 soll pro Cluster beziehungsweise virtuellem Rechenzentrum ein Prometheus HA Paar betrieben werden. Da es jedoch auch Sonderfälle gibt, in denen auf Kundenwunsch hin ein Rechenzentrum nur eine VM beinhaltet, ist es in einem solchen Fall nicht praktikabel, hier ebenfalls ein HA Setup umzusetzen. Daher wird diese Anforderung mit mittlerer Priorität eingestuft (Experte A 2021; Experte B 2021).

ID	QA3	**Prio**	Mittel
Titel	Ein Prometheus HA Paar pro Cluster oder Einheit		
Herkunft	Experteninterviews		
Beschreibung			
In jedem Cluster beziehungsweise in jeder Einheit soll ein Prometheus HA Paar betrieben werden.			
Begründung			
Um die Metriken ausfallsicher einsammeln zu können, reicht es nicht aus, eine einzelne Prometheus Instanz zu betreiben.			
Abnahmekriterium			
In jeder Einheit, die überwacht werden soll, sei es ein Cluster oder ein virtuelles Rechenzentrum, wird ein Prometheus HA Paar betrieben.			

Tabelle 4.14: QA3 – Ein Prometheus HA Paar pro Cluster oder Einheit

4.4.4 ~99% Verfügbarkeit

Da das System unter anderem lückenlos Daten sammeln soll, wie in Unterabschnitt 4.4.7 beschrieben, und diese Daten in Zukunft auch für Kunden relevant werden können (Unterabschnitt 4.2.2), ist eine hohe Verfügbarkeit der Monitoring Lösung von hoher Priorität (Experte A 2021; Experte B 2021).

ID	QA4	**Prio**	Hoch
Titel	~99% Verfügbarkeit		
Herkunft	Experteninterviews		

Beschreibung
Das System soll eine Verfügbarkeit von ~99% oder höher aufweisen.

Begründung
Um eine lückenlose Überwachung der Systeme zu gewährleisten, muss die Monitoring Lösung eine hohe Verfügbarkeit aufweisen.

Abnahmekriterium
Die Systemarchitektur berücksichtigt die Anforderungen an die Verfügbarkeit.

Tabelle 4.15: QA4 – ~99% Verfügbarkeit

4.4.5 Verschlüsselte Kommunikation der Komponenten

Weil die Daten, welche erhoben werden, unter Umständen auch dem Datenschutz unterliegen oder kritische Informationen zu Systemen beinhalten können, soll die Kommunikation der Komponenten verschlüsselt stattfinden. Da die Komponenten jedoch vorerst nur in der eigenen Infrastruktur betrieben werden und von Seiten des Anbieters IONOS zugesichert wird, dass die Kommunikation in den virtuellen Rechenzentren getrennt von der anderer Kunden abläuft, ist diese Anforderung mit mittlerer Priorität eingestuft (Experte A 2021).

ID	QA5	**Prio**	Mittel
Titel	Verschlüsselte Kommunikation der Komponenten		
Herkunft	Experteninterviews		

Beschreibung
Die Komponenten der Monitoring Lösung sollen verschlüsselt miteinander kommunizieren.

Begründung
Da die gesammelten Daten unter Umständen auch datenschutzrelevant sein können, wenn es sich zum Beispiel um Kundensysteme handelt, soll die Kommunikation verschlüsselt stattfinden.

Abnahmekriterium
Die Kommunikation unter den Monitoring Komponenten läuft über eine verschlüsselte Verbindung ab.

Tabelle 4.16: QA5 – Verschlüsselte Kommunikation der Komponenten

4.4.6 Verschlüsselter S3 Speicher

Da die historischen Metriken sensible Daten enthalten können, sollen diese nur verschlüsselt abgelegt werden. Daher wird diese Anforderung mit hoher Priorität eingestuft (Experte A 2021; Experte B 2021).

ID	QA6	**Prio**	Hoch
Titel	Verschlüsselter S3 Speicher		
Herkunft	Experteninterviews		

Beschreibung
Der IONOS S3 Speicher muss die gespeicherten Daten verschlüsselt ablegen.

Begründung
Aus Datenschutzgründen sollen die Daten verschlüsselt gespeichert werden.

Abnahmekriterium
Gespeicherte Daten werden verschlüsselt abgelegt.

Tabelle 4.17: QA6 – Verschlüsselter S3 Speicher

4.4.7 Lückenloses einsammeln von Metriken

Damit bei Problemen Rückschlüsse auf deren Ursprung gezogen werden können, ist es erforderlich, Metriken zeitlich lückenlos zu sammeln. Daher wird diese Anforderung mit hoher Priorität eingestuft (Experte A 2021; Experte B 2021).

ID	QA7	**Prio**	Hoch
Titel	Lückenloses sammeln von Metriken		
Herkunft	Experteninterviews		

Beschreibung
Um gezielt auf Probleme reagieren zu können, soll die Monitoring Lösung zeitlich lückenlos Metriken sammeln.

Begründung
Um Probleme isolieren zu können, müssen genügend Daten über die überwachten Systeme vorliegen.

Abnahmekriterium
Metriken werden lückenlos gesammelt.

Tabelle 4.18: QA7 – Lückenloses sammeln von Metriken

4.4.8 Zeitnahe Benachrichtigung über Vorfälle

Um Reaktionszeiten zu verkürzen, sollen Benachrichtigungen so schnell wie möglich versendet werden. Da dies eine der wichtigsten Funktionen der Monitoring Lösung ist, wird diese Anforderung mit hoher Priorität eingestuft (Experte A 2021; Experte B 2021).

ID	QA8	**Prio**	Hoch
Titel	Zeitnahe Benachrichtigung über Vorfälle		
Herkunft	Experteninterviews		

Beschreibung
Um schnell auf Probleme reagieren zu können, soll die Monitoring Lösung bei Zwischenfällen zeitnah Benachrichtigungen verschicken.

Begründung
Benachrichtigungen über Vorfälle müssen zeitnah versandt werden, um Reaktionszeiten zu verkürzen.

Abnahmekriterium
Benachrichtigungen werden zeitnah versandt, sobald ein Problem auftritt.

Tabelle 4.19: QA8 – Zeitnahe Benachrichtigung über Vorfälle

4.4.9 Übersichtliche Dashboards

Gerade für die interne IT-Abteilung ist es wichtig, den Überblick über die Systeme zu wahren, daher wird diese Anforderung mit hoher Priorität eingestuft (Experte A 2021).

ID	QA9	**Prio**	Hoch
Titel	Übersichtliche Dashboards		
Herkunft	Experteninterviews		

Beschreibung
Die Dashboards der Monitoring Lösung sollen übersichtlich gestaltet sein.

Begründung
Um auf einen Blick Probleme oder den generellen Status der Systeme feststellen zu können, sollen die Dashboards übersichtlich gestaltet sein.

Abnahmekriterium
Dashboards sind nicht überladen und bieten grundlegende Informationen sowie relevante Details.

Tabelle 4.20: QA9 – Übersichtliche Dashboards

Kapitel 5

Systemarchitektur

5.1 Analyse Ist-Zustand

5.1.1 Nomad Cluster

Abbildung 5.1 zeigt das Verteilungsdiagramm des Nomad Clusters. Zu sehen sind die momentan für den Betrieb der internen Dienste verwendeten Systeme. Die Firewall beziehungsweise Router VM, auf der die Open Source Firewall IPFire[1] betrieben wird, spannt im LAN 1 das Netzwerk des Nomad Clusters auf. Außerdem sind hier VPN-Verbindungen definiert, um auf die Hosts hinter der Firewall zugreifen zu können.

Zusätzlich sind hier die Intrexx VMs zu sehen, welche unabhängig vom Nomad Cluster agieren und lediglich an die Solr VM angebunden sind, um den hier betriebenen Solr Suchserver[2] nutzen zu können.

[1] https://www.ipfire.org/
[2] https://solr.apache.org/

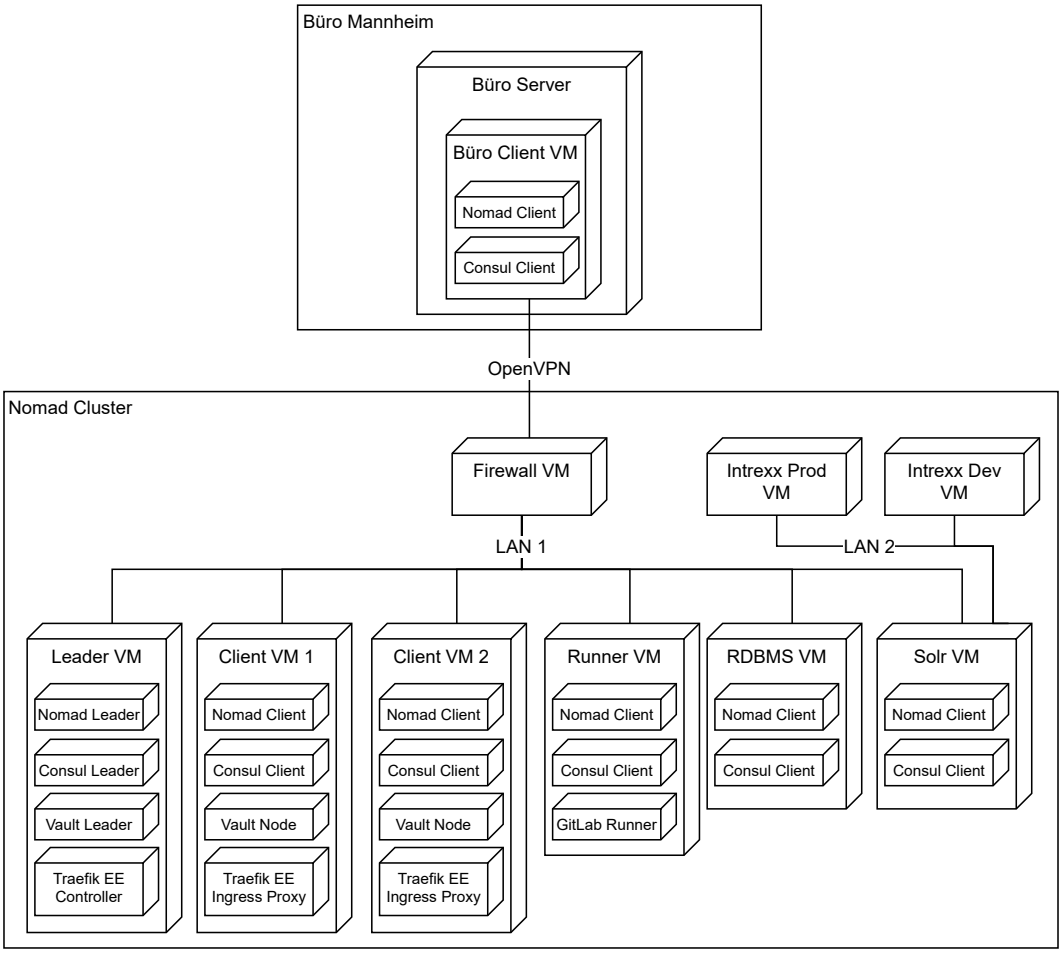

Abbildung 5.1: Verteilungsdiagramm des Nomad Clusters.

Leader VM

Auf der *Leader VM* läuft die nativ installierte Nomad Leader Instanz, welche die Jobs auf die verschiedenen Clients verteilt und eine Benutzeroberfläche für die Administration des Clusters zur Verfügung stellt. Außerdem sind hier die Leader Instanzen von Consul und Vault sowie der Controller des Traefik Enterprise Proxies installiert. Hier laufen keine Nomad Jobs, da diese VM lediglich für die Administration und Bereitstellung der zentralen Dienste im Cluster vorgesehen ist. Als Betriebssystem ist CentOS Linux Version 7 installiert.

Client VM 1 und Client VM 2

Diese beiden VMs sind die primären Nomad Clients, auf welche der Großteil der Nomad Jobs verteilt wird. Neben dem nativ installierten und im Client-Modus lau-

fenden Nomad Service, werden hier auch jeweils Replikationen von Vault als nativer Linux Service betrieben. Zusätzlich laufen hier auch noch native Consul Services im Client-Modus sowie die Ingress Proxy Services des Traefik Enterprise Proxies. Als Betriebssystem ist CentOS Linux Version 8 installiert.

Runner VM

Diese VM wird in erster Linie für die von GitLab für CI/CD Pipelines benötigten Runner verwendet, weshalb sie die meiste Rechenleistung im Cluster aufweist. Damit GitLab die Ressourcen der VM für die angestoßenen Pipelines nutzen kann, ist hier ein GitLab Runner installiert, welcher die Ablaufumgebung für die Pipelines zur Verfügung stellt (GitLab 2021c). Außerdem läuft hier ein nativer Nomad sowie Consul Service im Client-Modus. Als Betriebssystem ist CentOS Linux Version 8 installiert.

RDBMS VM

Diese VM wird vorrangig für den Betrieb der verschiedenen Datenbanken im Cluster verwendet. Daher sind hier in der Konfiguration des installierten Nomad Client Services verschiedene Host Volumes definiert, um die Daten der Datenbankcontainer zu persistieren. Außerdem läuft auch hier ein Consul Service im Client-Modus. Als Betriebssystem ist CentOS Linux Version 8 installiert.

Solr VM

Diese VM wird hauptsächlich für den Betrieb des Solr Suchservers im Cluster verwendet und dient außerdem als Test-Client für neue Nomad Jobs. Auch hier läuft ein Nomad und Consul Service im Client-Modus. Als Betriebssystem ist CentOS Linux Version 8 installiert.

5.1.2 Büro Client VM

Diese VM wird auf einem Server im Büro in Mannheim betrieben. Hier laufen Nomad und Consul Services im Client-Modus, außerdem ist die VM per OpenVPN-

Verbindung and das Nomad Cluster angebunden. Als Betriebssystem ist CentOS Linux Version 8 installiert.

5.1.3 Intrexx VMs

Die beiden übrigen Systeme *Intrexx Prod VM* und *Intrexx Dev VM*, welche ihren eigenen Internetzugang besitzen da diese selbst Dienste im Internet bereitstellen, sind über LAN 2 an die *Solr VM* angeschlossen, um den Solr Suchserver für die hier installierten Portal Server der Low-Code Lösung Intrexx[3] zur Verfügung zu stellen. Intrexx ist ein grafisches Fullstack Framework zur Entwicklung von Java Webapplikationen. Aus den grafisch kombinierten Elementen wird der Quellcode des Frontends und des Backends generiert, außerdem können auch benutzerdefinierte serverseitige Groovy Skripts ausgeführt werden. Als Betriebssystem ist jeweils Ubuntu Server 20.04 LTS installiert. Außerdem laufen hier jeweils eigenständige Traefik Proxy[4] Instanzen sowie Datenbanken für die Webapplikationen.

5.1.4 Dienste

Im Folgenden wird kurz darauf eingegangen, wie die verschiedenen Dienste, welche in Abbildung 5.1 zu sehen sind, verwendet werden.

Nomad

Wie schon im Grundlagenkapitel beschrieben, handelt es sich bei Nomad um eine leichtgewichtige Lösung zur Orchestrierung eines Clusters, in dem Container-Workloads, aber auch andere Jobs ausgeführt werden können.

Alle wesentlichen firmeninternen Tools werden in diesem Cluster betrieben. Dazu gehören unter anderem GitLab[5], LDAP und der intern verwendete Passwortmanager Passbolt[6] wie in Abbildung 5.2 zu sehen. Momentan läuft das Cluster auf Nomad Version 0.11.1.

[3] https://www.intrexx.com/
[4] https://doc.traefik.io/traefik/
[5] https://about.gitlab.com/
[6] https://www.passbolt.com/

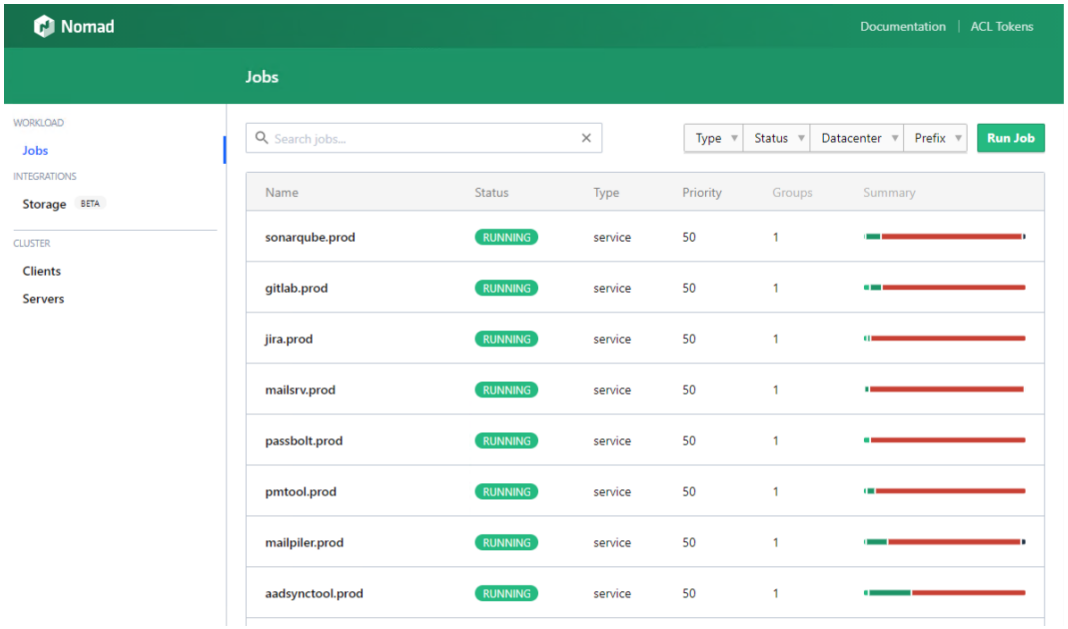

Abbildung 5.2: Screenshot der Nomad Weboberfläche, zu sehen ist ein Teil der aktuell laufenden Jobs.

Consul

Da alle Nomad Clients ebenfalls als Consul Clients agieren, ermöglicht dies die Kommunikation der verschiedenen Container in den jeweiligen Nomad Jobs untereinander. So kann zum Beispiel innerhalb der Nomad Jobs die Adresse sowie der Port anderer bei Consul registrierter Services innerhalb von Templates für Konfigurationen abgefragt werden, wie in Listing 5.1 zu sehen. Diese sogenannten Consul Templates werden beim Ausführen der jeweiligen Nomad Jobs mit den zugehörigen Werten befüllt und in der angegebenen Datei gespeichert. Momentan läuft das Consul Cluster auf Version 1.7.3 (HashiCorp 2021c).

```
- targets:
  - "{{ range service "prometheus-0-sidecar-grpc-dc1" }}{{ .Address }}:{{ .Port
      }}{{ end }}"
  - "{{ range service "prometheus-1-sidecar-grpc-dc1" }}{{ .Address }}:{{ .Port
      }}{{ end }}"
```

Listing 5.1: Beispiel für die Abfrage von Adressen und Ports innerhalb der Thanos Querier Konfigurationsvorlage mithilfe von Consul Templates.

Vault

Mithilfe von Consul Templates können innerhalb einer Konfigurationsvorlage eines Nomad Jobs ebenfalls Secrets von Vault abgefragt werden, wie in Listing 5.2 zu sehen. Vault wird in der Version 1.4.1 betrieben (HashiCorp 2021n).

```
type: "S3"
config:
  bucket: "metrics"
  endpoint: "s3-de-central.profitbricks.com"
  insecure: false
  signature_version2: true
  access_key: "{{ with secret "com.example/s3" }}{{ .Data.data.id }}{{ end }}"
  secret_key: "{{ with secret "com.example/s3" }}{{ .Data.data.secret }}{{ end }}"
  sse_config:
    type: "SSE-S3"
```

Listing 5.2: Beispiel für die Abfrage von Secrets von Vault innerhalb der Thanos Sidecar Konfigurationsvorlage mithilfe von Consul Templates.

Traefik Enterprise Edition

Den über einen Nomad Job bei Consul registrierten Services können Tags zugeordnet werden. Diese Tags werden von Traefik über Consul ausgelesen und eine entsprechende dynamische Konfiguration erstellt. Somit ist es nicht nötig, die Traefik Konfiguration von Hand anzupassen, wenn ein neuer Dienst mithilfe von Traefik über das Internet verfügbar gemacht werden soll.

```
service {
  name = "my-service"
  port = "my-port-name"
  tags = [
    "traefikv2.enable=true",
    "traefikv2.http.routers.my-example.rule=Host(`my.example.com`)",
    "traefikv2.http.routers.my-example.entrypoints=websecure",
    "traefikv2.http.routers.my-example.middlewares=my-middleware@traefikee"
  ]
}
```

Listing 5.3: Beispiel für eine Service Definition innerhalb eines Nomad Jobs unter Verwendung von Consul Tags für Traefik EE.

Listing 5.3 zeigt ein Beispiel für eine Service Definition innerhalb eines Nomad Jobs. Anhand der Namen von Port und Service innerhalb der Service Definition kann Traefik die Adresse und den Port des Services dynamisch von Consul abfragen und anhand der Tags die entsprechende Konfiguration vornehmen. In diesem Beispiel soll der Service unter der Domain *my.example.com* über den *websecure*

(Standardmäßig Port 443 mit TLS) Entrypoint angeboten werden. Zusätzlich ist noch eine Middleware vor den Service geschalten. Eine solche Middleware kann beispielsweise ein HTTP BasicAuth sein, um die Webseite mit einem Passwort zu schützen. Traefik EE wird in der Version 2.3 betrieben.

5.2 Analyse der Integrations- und Betriebsmöglichkeiten

Im Rahmen der Anforderungsanalyse haben sich einige Randbedingungen ergeben, darunter beispielsweise die Bedingung, dass der Großteil der Komponenten der Monitoring Lösung im eigenen Nomad Cluster, beziehungsweise in IONOS Rechenzentren, betrieben werden soll. Zu den Komponenten, welche im Nomad Cluster betrieben werden sollen, gehören die Prometheus Instanzen mit den zugehörigen Thanos Sidecars und Querieren sowie die Alertmanager Instanzen. Zu den Komponenten welche an übergeordneter Stelle, außerhalb des Nomad Clusters, betrieben werden sollen, gehören dagegen beispielsweise Grafana sowie das Thanos Query Frontend und das Thanos Store Gateway. Außerdem kam während der Experteninterviews die Möglichkeit hinzu, bestimmte Komponenten mithilfe des Managed Kubernetes Angebots (Kubernetes als IaaS Lösung) von IONOS zu betreiben (Experte B 2021). Für die langfristige Speicherung der Metriken soll das S3 Angebot von IONOS genutzt werden. Beide Lösungen können direkt über den IONOS DCD in das bestehende virtuelle Rechenzentrum integriert, beziehungsweise darüber administriert werden.

Prinzipiell bestehen folgende Möglichkeiten für den Betrieb der Komponenten:

- Als Nomad Job im bestehenden Cluster
- Im Managed Kubernetes von IONOS
- Als nativ installierter Dienst oder Container auf einer dedizierten VM

5.2.1 Auswahl- und Bewertungskriterien

Folgende Auswahl- und Bewertungskriterien werden für die Analyse beziehungsweise den Vergleich festgelegt:

1. Integrationsfähigkeit in die bestehende Infrastruktur

2. Personalaufwand für Betrieb und Nutzung
3. Ausfallsicherheit der Lösung

Alle drei Kriterien werden gleich gewichtet, da sie aus wirtschaftlicher Sicht eng miteinander verknüpft sind. So wäre zum Beispiel eine sehr gute Ausfallsicherheit und Integrationsfähigkeit in Verbindung mit einem hohen Personalaufwand nicht wünschenswert, gleiches gilt für den umgekehrten Fall.

5.2.2 Vergleich der Betriebsmöglichkeiten

Im Folgenden werden die Betriebsmöglichkeiten in Tabelle 5.1 gegenübergestellt. Die Bewertung findet mithilfe von Harvey Balls statt, ein vollständig gefüllter Ball entspricht also dem besten und ein leerer dem schlechtesten Ergebnis, wobei die Füllung immer in Viertel Schritten erfolgt. Zum Beispiel entspricht ein vollständig gefüllter Ball bei Personalaufwand dem besten Ergebnis, also den geringsten Kosten.

	Nomad Cluster	Managed Kubernetes	Dedizierte VM
1. Integrationsfähigkeit	●	◔	●
2. Personalaufwand	●	◑	◕
3. Ausfallsicherheit	◕	●	◔

Tabelle 5.1: Vergleich der Betriebsmöglichkeiten.

Da das Nomad Cluster integraler Bestandteil der Infrastruktur ist, alle verantwortlichen Mitarbeiter ausreichend Erfahrung mit den Systemen haben und außerdem alle internen Dienste in diesem Cluster betrieben werden, erhält das Nomad Cluster die insgesamt beste Bewertung.

Darauf folgt die Verwendung einer dedizierten VM. Die Integrationsfähigkeit ist aufgrund der Tatsache, dass jegliche benötigten Systemeinstellungen vorgenommen werden können, sehr gut. Auch der Personalaufwand ist recht gering, da das Einrichten einer VM eine alltägliche Aufgabe des Operations Teams darstellt. Einzig die Ausfallsicherheit ist hier nicht optimal, da es sich um ein einzelnes System handelt, welches über keinerlei Redundanz verfügt. Trotzdem wird nicht die schlechteste Wertung vergeben, da die IONOS Infrastruktur nach Erfahrung des Operations Teams sehr zuverlässig ist (Experte A 2021).

Das Managed Kubernetes Angebot landet auf dem letzten Platz, weil aufgrund mangelnder Möglichkeiten zur individuellen Konfiguration der Komponenten keine gute Integrationsfähigkeit gegeben ist. Außerdem ist keine flächendeckend gute Kenntnis von Kubernetes im Operations Team vorhanden, weshalb der Personalaufwand im Vergleich zu den anderen beiden Lösungen zumindest anfänglich recht hoch ausfallen würde. Auch die Administration einer zusätzlichen Technologie bedeutet einen größeren Personalaufwand. Die Ausfallsicherheit ist hingegen ausgezeichnet, da Updates sowie die Handhabung der verschiedenen Kubernetes Komponenten größtenteils von IONOS übernommen werden.

5.2.3 Auswahl der Betriebsmöglichkeiten

Generell ist es nötig, die darstellenden (z.B. Grafana) beziehungsweise abfragenden (z.B. Prometheus) Komponenten getrennt voneinander zu betreiben. Die Trennung in zwei Schichten ist deshalb erforderlich, weil die Monitoring Lösung in Zukunft noch erweitert werden soll, wenn zum Beispiel zusätzlich noch andere Systeme oder virtuelle Rechenzentren angebunden werden. In einem solchen Fall würden im zusätzlich anzubindenden virtuellen Rechenzentrum ebenfalls Prometheus und Alertmanager Instanzen betrieben werden, um mithilfe dieser Instanzen die Metriken der im Rechenzentrum vorhandenen Systeme einzusammeln. Die darstellenden Komponenten müssen also bei einem solchen Setup an übergeordneter Stelle stehen, um Daten von allen angebundenen Rechenzentren oder Clustern abfragen zu können. Ein Beispiel hierfür ist in Abbildung 2.16 in Unterabschnitt 2.7.2 zu finden.

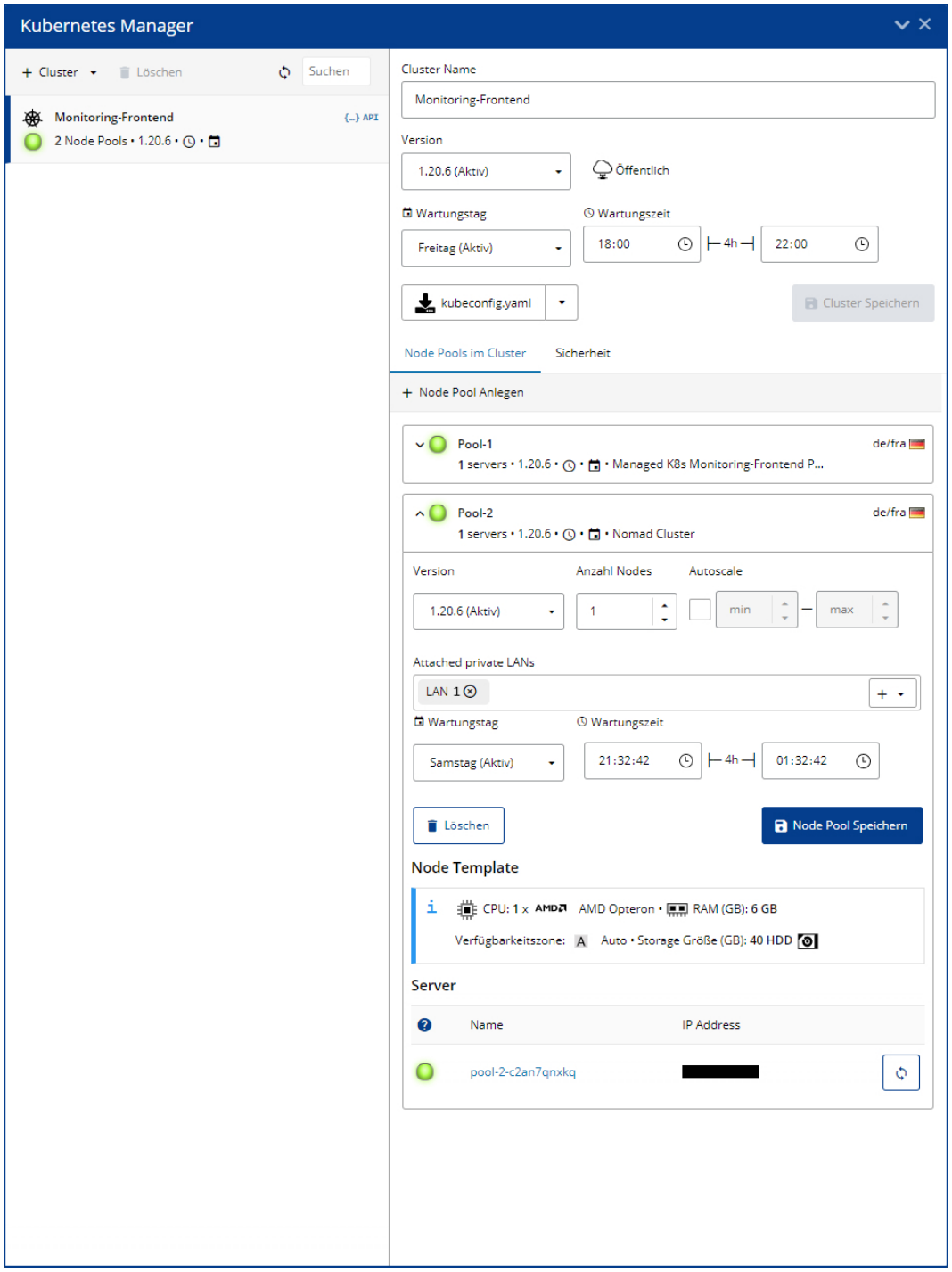

Abbildung 5.3: Screenshot des Kubernetes Managers im IONOS DCD.

Erste Versuche mit dem Managed Kubernetes Angebot von IONOS haben gezeigt, dass sich das Kubernetes Cluster nicht ohne weiteres an das bestehende Nomad Cluster anbinden lässt. Eine Verbindung der beiden Cluster über VPN ist jedoch nötig, da Thanos sowie Prometheus und Alertmanager zurzeit nur experimentelle

oder gar keine Unterstützung für TLS sowie die Authentifizierung der Schnittstellen bieten (Prometheus 2021j; Thanos 2021b).

Wie in Abbildung 5.3 zu erkennen, lassen sich über das Managed Kubernetes lediglich einzelne Cluster sowie deren jeweilige Node Pools anlegen. Das Cluster stellt hier die Control Plane dar, welche komplett von IONOS verwaltet wird. Die Einzelnen Pools sind jeweils einem virtuellen Rechenzentrum zugeordnet, in dem die jeweiligen Worker Nodes vollautomatisch erstellt werden. Auf diese automatisch erstellten VMs kann jedoch weder über die Konsole im IONOS DCD noch über SSH zugegriffen werden. Es ist zwar möglich, private LANs an Pools anzubinden, wie in Abbildung 5.3 zu sehen, dies gilt dann jedoch nur für den jeweiligen Pool. Außerdem könnten Pools in anderen Regionen so nicht angebunden werden, ohne noch eine zusätzliche Router VM zu verwenden. So müssten alle Pools entweder direkt über LAN an das Nomad Cluster sowie eventuelle andere virtuelle Rechenzentren angebunden oder pro Pool eine Router VM für die VPN-Verbindungen in die jeweiligen virtuellen Rechenzentren und Cluster verwendet werden.

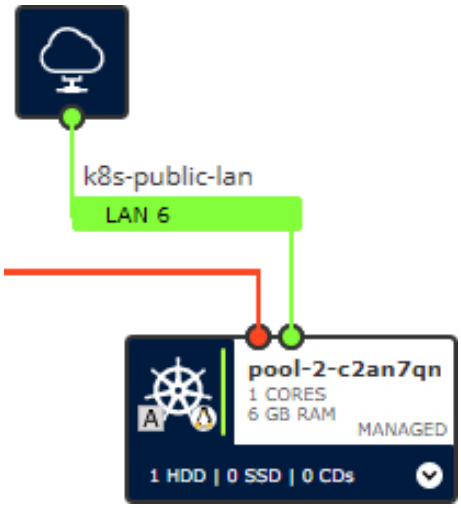

Abbildung 5.4: Screenshot einer automatisch erstellten Kubernetes Worker Node im Nomad Cluster im IONOS DCD.

Die Worker Node *pool-2-c2an7qn* in Abbildung 5.4 ist an LAN 1 des Nomad Clusters aus Abbildung 5.1 angebunden und besitzt einen eigenen Internetzugang. Die Firewall und weitere Netzwerkkonfigurationen können ebenfalls nicht bearbeitet werden.

Aufgrund der genannten Einschränkungen stellt sich das Managed Kubernetes Angebot für den geplanten Einsatzzweck als Monitoring Cluster wie in Abbildung 2.16 dargestellt, als ungeeignet heraus. Dies ist teilweise auch den experimentellen oder

nicht vorhandenen Sicherheitsfunktionen der einzelnen Monitoring Komponenten geschuldet, weshalb der Einsatz eines VPN erforderlich wird, um eine sichere Kommunikation gewährleisten zu können.

Damit bleibt in Hinblick auf das Monitoring Cluster die Möglichkeit der Verwendung des existierenden Nomad Clusters oder einer dedizierten VM. Da die darstellenden Komponenten jedoch aus anfänglich genannten Gründen getrennt von den abfragenden Komponenten im bestehenden Nomad Cluster betrieben werden sollen, ist eine dedizierte VM die letzte verbleibende Option für die darstellenden Komponenten.

Da ein Ausfall der Darstellung lediglich bedeutet, dass die Weboberflächen zur Darstellung von Dashboards und zum Absetzen von PromQL Abfragen nicht verfügbar sind, ist diese Lösung jedoch vertretbar. Metriken werden weiterhin von den Prometheus Instanzen gesammelt und bei Vorfällen Alerts über die Alertmanager versendet.

5.3 Systementwurf

5.3.1 Verteilung der Komponenten

Da das Managed Kubernetes Angebot von IONOS wie in Abschnitt 5.2 beschrieben nicht geeignet ist, wird der ursprüngliche Plan, alle Komponenten als HA Setup zu betreiben dahingehend aufgeweicht, dass das Monitoring Cluster vorerst nur als einzelne VM in einer anderen Region realisiert wird. Die Monitoring Komponenten laufen wiederum als Docker Container auf eben dieser VM. Damit ist das virtuelle Rechenzentrum de facto vorerst kein Cluster. Die Benennung bleibt jedoch mit Blick auf die mögliche weitere Entwicklung bestehen.

Wie in Abbildung 5.5 zu erkennen, werden das Monitoring Cluster und das Nomad Cluster über eine OpenVPN-Verbindung miteinander verbunden. Dies hat den Hintergrund, dass die einzelnen Monitoring Komponenten zum Zeitpunkt des Entwurfs keine flächendeckende Unterstützung für die Authentifizierung der Schnittstellen oder die verschlüsselte Kommunikation bieten (Prometheus 2021j; Thanos 2021b).

Der Thanos Querier im Monitoring Cluster kommuniziert über gRPC mit den Thanos Querier Instanzen im Nomad Cluster, um Metriken abfragen zu können. In Abbildung 5.5 fehlt der Thanos Querier im Nomad Cluster, da dieser Container

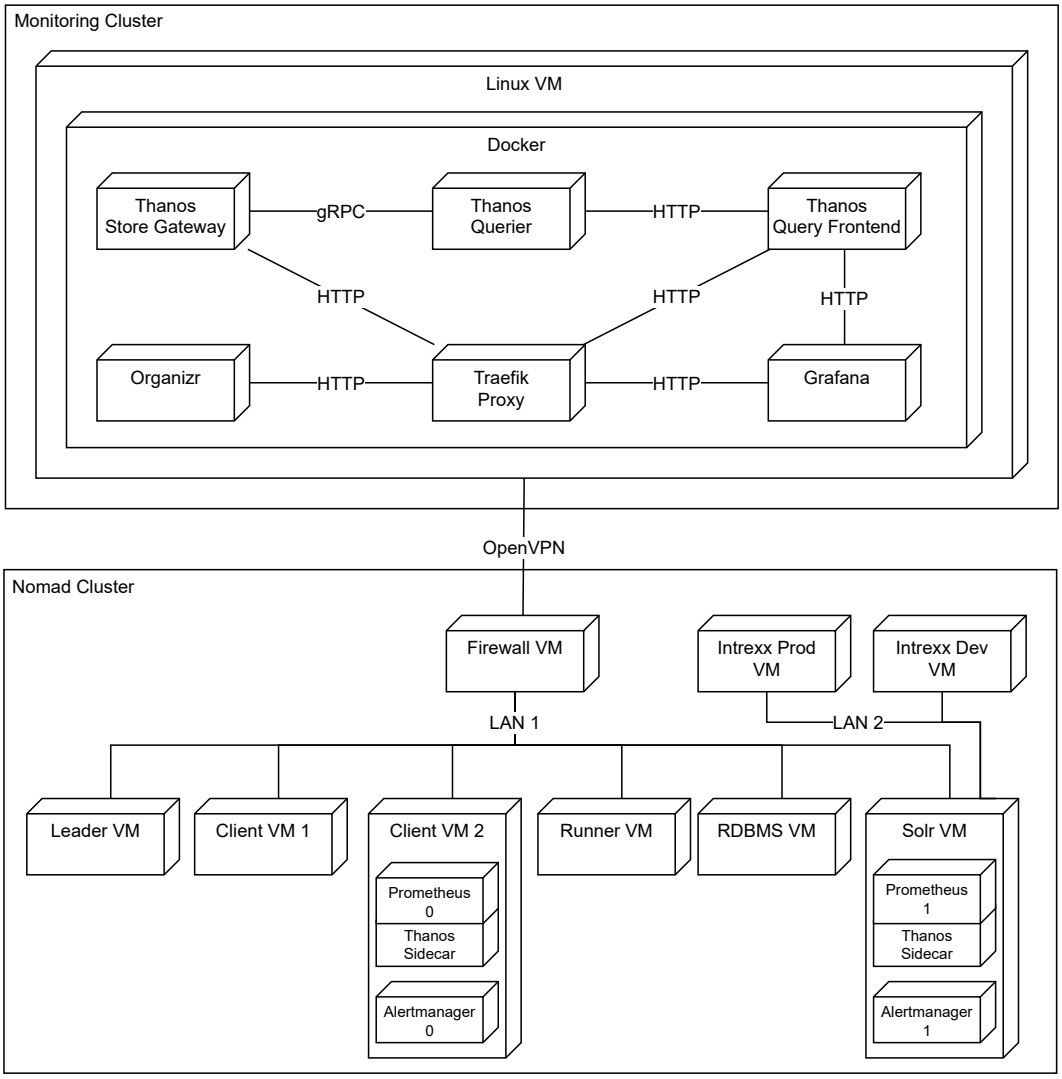

Abbildung 5.5: Verteilungsdiagramm für die statisch verteilten Monitoring Komponenten.

zustandslos ist und damit dynamisch auf jeden Client im Nomad Cluster verteilt betrieben werden kann. Auch Replikate dieser Komponente, um eine gewisse Ausfallsicherheit zu gewährleisten, sind möglich. Die Kommunikation der Thanos Sidecars und des Thanos Store Gateways mit dem IONOS S3 Speicher wird hier nicht weiter betrachtet, da diese ohnehin über das öffentliche Netz unter Verwendung von Hypertext Transfer Protocol Secure (HTTPS) stattfindet. Außerdem kommt auf der VM im Monitoring Cluster ein Traefik Proxy in Verbindung mit Organizr zum Einsatz, um eine zentrale Weboberfläche für alle Komponenten auf der VM zur Verfügung zu stellen.

Organizr wird deshalb gewählt, weil hier eine LDAP Anbindung möglich ist, um Benutzer zu authentifizieren. In Verbindung mit der *ForwardAuth* Middleware von

Traefik ist es so möglich, mithilfe von Organizr und dem angebundenen LDAP eine Authentifizierung für alle Weboberflächen zu implementieren, ohne dass diese eine separate Authentifizierung benötigen oder zur Verfügung stellen. Im Detail wird bei jeder Anfrage an die speziellen Weboberflächen, beispielsweise der des Thanos Query Frontends, über den Traefik Proxy zuerst eine Anfrage von Traefik an Organizr gestellt, um festzustellen, ob der anfragende Nutzer über Organizr authentifiziert ist. Wenn dies der Fall ist, wird die Anfrage an die jeweilige Weboberfläche weitergeleitet, und der Nutzer erhält die Antwort über den Traefik Proxy (HalianElf 2019; HalianElf 2021; Containous 2021b).

5.3.2 Konfigurationsmanagement

Da die Konfiguration einiger Komponenten der Monitoring Lösung bedingt durch deren Verwendung potenziell öfter wechselt, wird es nötig, eine zentrale Steuerung dieser Konfigurationen zu implementieren. Im Vordergrund stehen hier vor allem die Prometheus Instanzen, da zum Beispiel für jede neue Alerting Regel oder jedes neue Target, von dem Metriken abgefragt werden sollen, die Konfigurationsdatei angepasst werden muss. Da die beiden geplanten Prometheus Instanzen als HA Paar betrieben werden sollen, ist es außerdem wichtig, dass beide Instanzen dieselbe Konfiguration verwenden.

Die Konfigurationen der anderen Komponenten sind hingegen eher statisch und ändern sich im Fall des Alertmanagers zum Beispiel nur, wenn eine neue Versandart oder eine neue Gruppierung für Benachrichtigungen verwendet werden soll.

Konkret können im Fall der im Nomad Cluster betriebenen Komponenten alle Möglichkeiten von Nomad in Verbindung mit GitLab zum Einsatz kommen. So wird ein gesondertes Repository für die Speicherung und Versionierung der Konfigurationsdateien angelegt und ein Nomad Job sowie eine GitLab CI/CD Pipeline eingerichtet, um die Konfigurationsdateien auf den jeweiligen Clients auszutauschen. Das erneute laden der Konfiguration wird von Nomad über das Senden eines *SIGHUP* Signals an die jeweiligen Prozesse im Container angestoßen.

So muss bei Änderungen der Konfiguration im Repository lediglich die CI/CD Pipeline gestartet werden, wodurch ein Nomad Job ausgeführt wird, welcher die neue Konfiguration ausrollt.

Kapitel 6

Machbarkeitsnachweis

6.1 Umsetzung des Systementwurfs

6.1.1 Umsetzung im Monitoring Cluster

Das virtuelle Rechenzentrum wird vollständig mithilfe von Terraform provisioniert und verwaltet. Hierfür wird in GitLab ein Repository mit einer CI/CD Pipeline angelegt, um so die virtuelle Infrastruktur sowie die Konfiguration der VM komplett über Terraform und Ansible Definitionen verwalten zu können. Abbildung 6.1 zeigt die CI/CD Pipeline zur Steuerung des Monitoring Clusters. Das virtuelle Rechenzentrum wird in der IONOS Region Berlin erstellt.

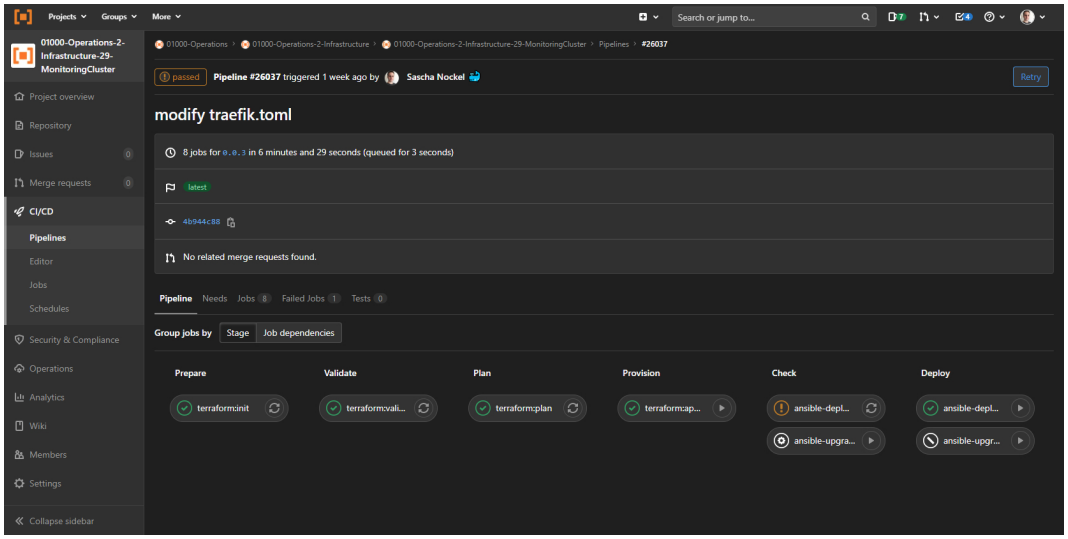

Abbildung 6.1: Screenshot der GitLab Deployment Pipeline unter Verwendung von Terraform und Ansible.

Folgende Stages sind in der Pipeline enthalten:

1. Prepare: Vorbereitung der Terraform Umgebung mit *terraform init*.
2. Validate: Validierung der Terraform Konfiguration mit *terraform validate*.
3. Plan: Erstellung des Ablaufplans für die Terraform Konfiguration zur Kontrolle vor dem Ausführen mit *terraform plan*.
4. Provision: Anwenden der Terraform Konfiguration mit *terraform apply*.
5. Check: Überprüfung der Ansible Playbooks mithilfe der *check* und *diff* Flags.
6. Deploy: Anwenden der Ansible Playbooks.

Für jedes neu erstellte Tag innerhalb des Repositories wird eine Pipeline erstellt. So können verschiedenen Versionen der Infrastruktur immer eindeutige Tags zugeordnet werden. Die Pipeline startet automatisch mit der ersten Stage und führt nacheinander alle Jobs bis zur *Provision* Stage aus. Die *Provision* Stage muss manuell gestartet werden, um eine Kontrolle vor dem Anwenden der Änderungen zu ermöglichen. Nach dem erfolgreichen Abschluss der *Provision* Stage wird das für Ansible generierte Host Inventory als Artefakt für die nächste Stage gespeichert. Die *Check* und *Deploy* Stages können erst nach dem erfolgreichen Ablauf der vorherigen Stages gestartet werden. Dies hat den Hintergrund, dass sich beispielsweise die IPs innerhalb des Ansible Host Inventories oder aber die Infrastruktur ändern könnte, falls Änderungen an der Terraform Konfiguration vorgenommen wurden. Da der Terraform State im Terraform HTTP Backend des Repositories gespeichert ist, verändert das mehrmalige Anwenden der Terraform Konfiguration die Infrastruktur nur, wenn Anpassungen vorgenommen wurden (GitLab 2021b).

Die Stages *Check* und *Deploy* beinhalten die Ansible Jobs. Diese sind in zwei Abzweigungen aufgeteilt, einerseits die *ansible-deploy* Jobs und andererseits die *ansible-upgrade* Jobs. Dies dient dazu, das Deployment der Anwendungen von generellen Wartungsaufgaben, wie dem Einspielen von Paketupdates, zu trennen. Jede der beiden Abzweigungen muss manuell gestartet werden, da nicht unbedingt immer Infrastruktur und Deployment gleichzeitig angepasst werden. In der *Check* Stage wird das jeweilige Ansible Playbook geprüft und die benötigten Änderungen am Deployment ermittelt. Nach Abschluss dieser Stage kann das Playbook über den manuellen Start des Jobs in der *Deploy* Stage angewendet werden.

Mithilfe dieser Deployment Pipeline wird die VM mit Terraform im virtuellen Rechenzentrum provisioniert, anschließend über Ansible konfiguriert und über OpenV-

PN mit dem Nomad Cluster verbunden. Die erforderlichen Zertifikate für die VPN-Verbindung sowie die Docker Compose Definition und die Konfigurationen der Container werden innerhalb des Pipeline Jobs mit Daten aus Vault und Consul befüllt, beziehungsweise aus Vault abgerufen und anschließend auf die VM übertragen. Abschließend wird die Docker Compose Definition durch Ansible auf der VM gestartet. Ein manuelles Einloggen auf die VM ist somit nicht nötig.

6.1.2 Umsetzung im Nomad Cluster

Für die Umsetzung des Systementwurfs innerhalb des Nomad Clusters wird ein Nomad Job erstellt. Hierfür wird ein gesondertes Repository mit Deployment Pipeline in GitLab verwendet. Die Deployment Pipeline beinhaltet eine *Deploy* Stage mit einem einzigen Job. Innerhalb dieses Pipeline Jobs wird der Nomad Job mithilfe des Nomad Terraform Providers an die Nomad API gesendet. Nomad evaluiert daraufhin die Job Spezifikation und verteilt die Komponenten auf das Nomad Cluster.

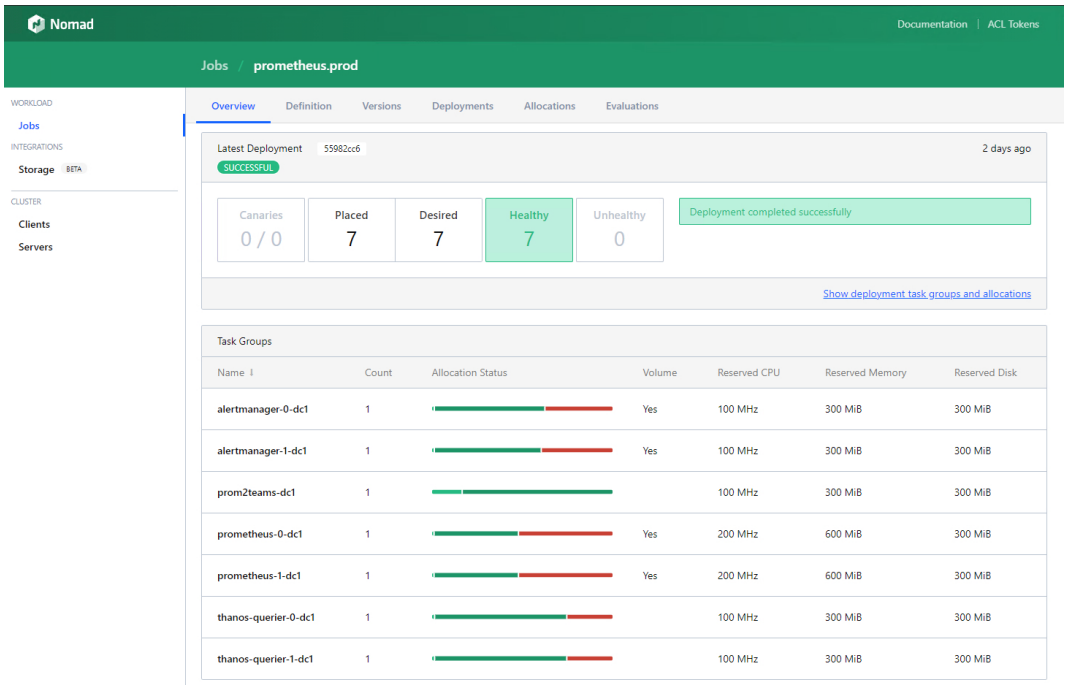

Abbildung 6.2: Screenshot der Task Gruppen innerhalb des Prometheus Nomad Jobs.

Abbildung 6.2 zeigt die Task Gruppen innerhalb des Nomad Jobs. Jede Task Gruppe wird von Nomad nach Möglichkeit auf einen separaten Client verteilt, sofern es im Nomad Job nicht anders definiert ist. Die Prometheus Task Gruppen beinhalten die Prometheus Instanzen sowie die jeweiligen Sidecars. Die restlichen Task

Gruppen beinhalten jeweils die Alertmanager sowie Thanos Querier Instanzen. Die *prom2teams-dc1* Task Gruppe beinhaltet eine Instanz des *prom2teams* Containers[1]. Dieser Container agiert als Proxy für die von den Alertmanager Instanzen per Webhook versendeten Benachrichtigungen und bringt diese vor dem Weiterleiten in das richtige Format für den Webhook Endpunkt von Microsoft Teams. Die Alertmanager und Prometheus Instanzen verwenden außerdem Host Volumes auf ihren jeweiligen Nomad Clients zur Persistierung der Daten. Alle anderen Komponenten sind zustandslos.

6.1.3 Zusammenspiel der Komponenten

Im Folgenden wird das Zusammenspiel der verschiedenen Komponenten beschrieben und erläutert. Dazu gehört die Abfragehierarchie für Metriken, der Ablauf für den Versand von Alerts sowie die Funktionsweise des Frontends der Monitoring Lösung.

Abfrage von Metriken

Abbildung 6.3 zeigt die Abfragehierarchie für Metriken. An der Spitze stehen Grafana und das Thanos Query Frontend, wobei Grafana die Schnittstelle des Thanos Query Frontends verwendet, um bei wiederholten Abfragen, die vom Query Frontend zwischengespeicherten Ergebnisse verwenden zu können. Hierdurch werden die Abfragezeiten verkürzt und Bandbreite gespart.

Das Thanos Query Frontend kann immer nur einen Downstream Querier verwenden, um Metriken abzufragen. Daher wird im Monitoring Cluster ein Thanos Querier betrieben, um die Schnittstellen der Querier im Nomad Cluster und die des Thanos Store Gateways im Monitoring Cluster zusammenzufassen.

Die Thanos Querier im Nomad Cluster kommunizieren jeweils mit den Thanos Sidecars der Prometheus Instanzen. Die Sidecars wiederum leiten Anfragen, die an sie gestellt werden, an die HTTP API der jeweiligen Prometheus Instanz weiter und laden gleichzeitig gesammelte Metriken in den IONOS S3 Speicher hoch.

Wenn nun eine PromQL Abfrage gestellt wird, geht diese zuerst ins Nomad Cluster bis hin zu den Prometheus Instanzen. Wenn die angefragten Daten hier nicht ver-

[1] https://github.com/idealista/prom2teams

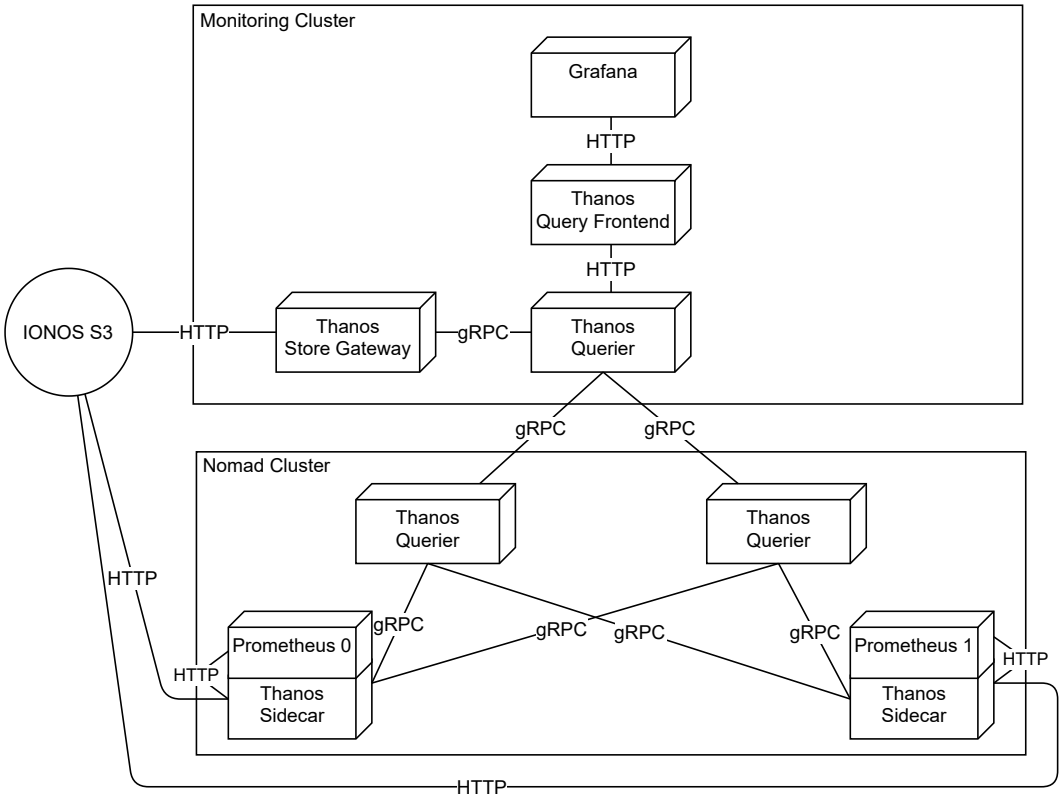

Abbildung 6.3: Abfragehierarchie für Metriken.

fügbar sind, weil diese zu weit zurück liegen, wird über das Thanos Store Gateway im IONOS S3 Speicher gesucht. So werden nur Daten aus dem S3 Speicher abgerufen, wenn diese nicht mehr in den Prometheus Instanzen vorhanden sind. Dadurch können zusätzliche Kosten, die durch das Abrufen von Daten aus dem S3 Speicher anfallen würden, gespart werden.

Die Prometheus Instanzen sind so konfiguriert, dass sie Metriken für sieben Tage lokal zwischenspeichern, da dies der relevanteste Zeitraum ist. Alle älteren Metriken werden bis zu 30 Tage im IONOS S3 gespeichert. Diese Konfiguration beruht auf den Erkenntnissen der Experteninterviews, in denen klar wurde, dass die Metriken einer Woche am relevantesten sind und eine Speicherung von historischen Metriken für 30 Tage vorerst ausreicht (Experte A 2021; Experte B 2021).

Versand von Alerts

Abbildung 6.4 zeigt die Kommunikation und die Verbindungen der Komponenten, welche am Versand von Alerts beteiligt sind. Die Prometheus Instanzen werten die

Alerting Regeln anhand ihrer lokalen Time Series Daten aus. Falls eine Alert Regel positiv ausgewertet wird, versenden die Prometheus Instanzen einen Alert an alle ihre registrierten Alertmanager. Diese doppelt versendeten Alerts werden von den Alertmanagern im Alertmanager Cluster dedupliziert, da alle Alertmanager über ein Gossip Protokoll untereinander synchronisiert werden.

Die Alerts werden nun über E-Mail sowie über den Prom2Teams Proxy Container versendet, welcher die Alerts im richtigen Format an den Webhook Endpunkt von Microsoft Teams weiterleitet.

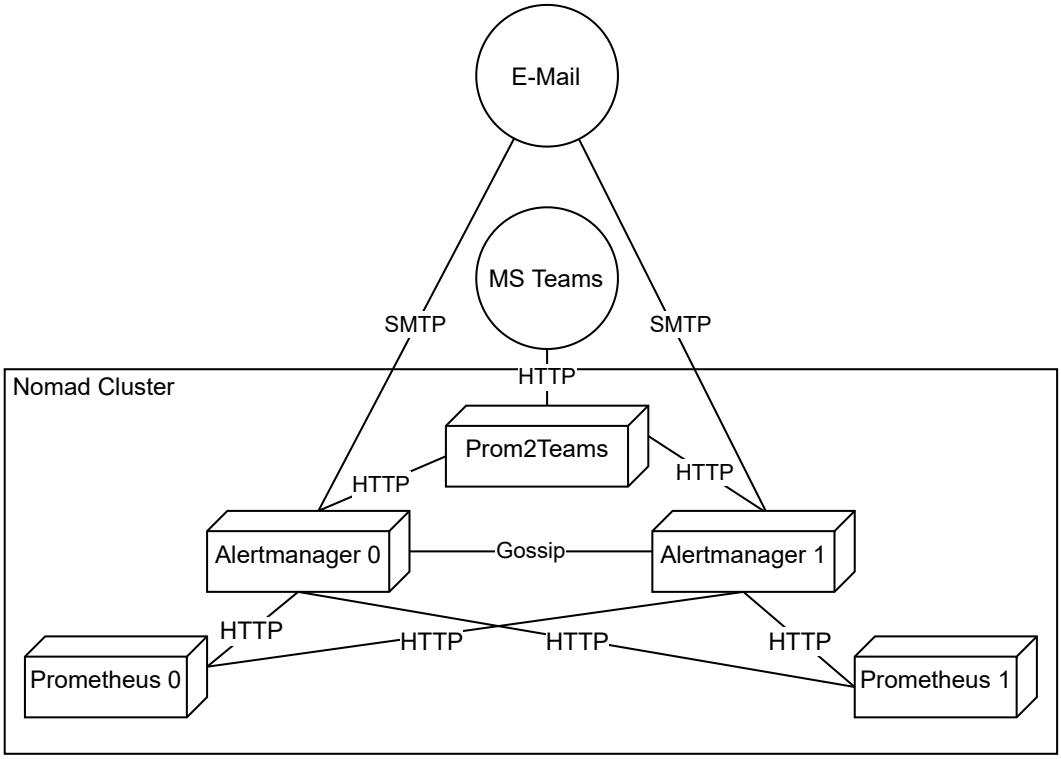

Abbildung 6.4: Zusammenhänge für den Versand von Alerts.

Monitoring Frontend

Auf der VM im Monitoring Cluster werden mithilfe des dort betriebenen Traefik Proxies die Weboberflächen der lokal betriebenen Container von Grafana, Thanos Query Frontend und Thanos Store Gateway bereitgestellt. Außerdem werden die Weboberflächen der Prometheus und Alertmanager Instanzen über die VPN-Verbindung zugänglich gemacht. So kann mithilfe von Organizr ein zentrales Frontend mit allen benötigten Komponenten bereitgestellt werden.

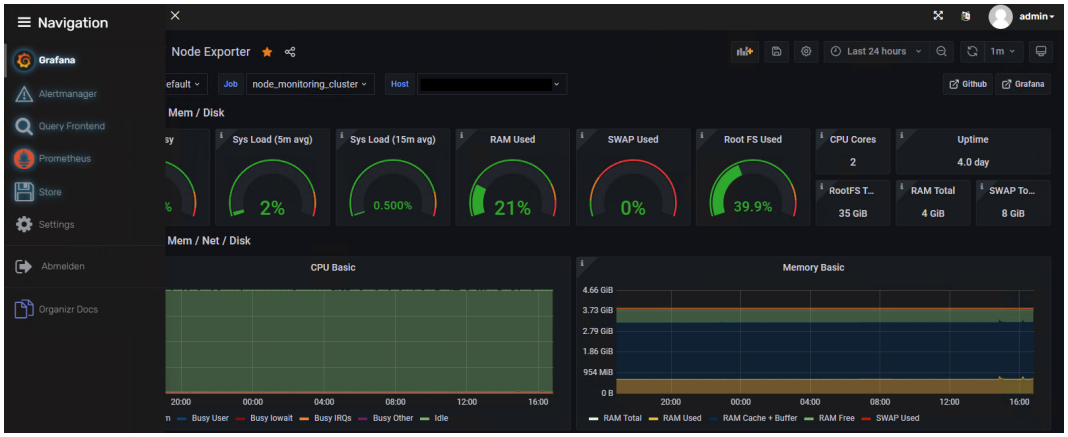

Abbildung 6.5: Screenshot des Monitoring Frontends.

Abbildung 6.5 zeigt das Frontend der VM im Monitoring Cluster. Die zu sehenden Komponenten werden über den lokalen Traefik Proxy über die Domain der VM bereitgestellt und mithilfe von Organizr zu einem Frontend zusammengefasst. Dies dient dazu, dass nicht für jede Weboberfläche ein eigener Browser Tab geöffnet werden muss.

Organizr und Grafana werden mit dem LDAP von bitExpert verbunden, um die Authentifizierung mithilfe der existierenden Benutzerkonten zu ermöglichen. Da Grafana eine eigene Benutzerverwaltung beinhaltet, wird dessen Weboberfläche nicht über Organizr authentifiziert.

6.1.4 Anbindung erster Systeme an das Monitoring

Um erste Daten für die Darstellung und Auswertung zu erhalten, werden folgende Systeme an das Monitoring angebunden:

- Traefik Instanzen der Intrexx VMs sowie der VM des Monitoring Clusters.

- Node Exporter auf den Intrexx VMs sowie der VM des Monitoring Clusters.
- Prometheus, Alertmanager und Thanos Instanzen im Nomad Cluster.
- Consul Exporter im Nomad Cluster.

Um die Abfrage von Metriken von den Traefik Instanzen zu ermöglichen, wird hier die Metriken Schnittstelle aktiviert (Containous 2021c). Die Node Exporter[2], welche Metriken der Hosts sammeln, stellen ebenfalls eine Metriken Schnittstelle bereit, welche ebenso über die Traefik Instanzen zur Verfügung gestellt wird. Beide Schnittstellen werden jeweils mit der HTTP BasicAuth Middleware von Traefik abgesichert (Containous 2021a). Außerdem wird in den Firewalls der VMs eine Verbindung auf die jeweiligen Ports nur für die IP des Gateways des Nomad Clusters erlaubt.

Die Prometheus, Alertmanager und Thanos Instanzen bieten bereits eine native Metriken Schnittstelle, die innerhalb des Nomad Clusters abgefragt werden kann. Außerdem wird im Nomad Cluster ein Consul Exporter[3] betrieben, welcher Daten von Consul abfragt und diese über seine Metriken Schnittstelle bereitstellt.

6.2 Definition grundlegender Alerting Regeln

Eine Prometheus Alert Regel hat im Wesentlichen fünf Bestandteile, *alert* für den Namen des Alerts, *expr* für die auszuwertende PromQL Abfrage, *for* zum festlegen des Zeitraums, für den die Regel zutreffen muss, um einen Alert auszulösen, *labels* zur Definition von Labels für den Alert und schließlich *annotations*, in denen zusätzliche Details vermerkt werden können.

Innerhalb einer Alert Regel können auch Laufzeitvariablen der Regel beziehungsweise von Prometheus verwendet werden, wie im *annotations* Bereich von Listing 6.1 zu sehen. In dieser Regel wird bei jeder Auswertung geprüft, ob ein Target den Wert 0 (nicht erreichbar) hat. Falls dies für 0 Minuten der Fall ist, also sofort bei Bekanntwerden eines nicht erreichbaren Targets, wird ein Alert an die registrierten Alertmanager gesendet. Vor dem Absenden werden die Variablen *value* (Ergebniswert der Abfrage) und *labels* von Prometheus mit den entsprechenden Werten befüllt.

[2] https://github.com/prometheus/node_exporter
[3] https://github.com/prometheus/consul_exporter

```
- alert: PrometheusTargetMissing
  expr: up == 0
  for: 0m
  labels:
    severity: critical
  annotations:
    summary: Prometheus target missing (instance {{ $labels.instance }})
    description: "A Prometheus target has disappeared. An exporter might be
        crashed.\n VALUE = {{ $value }}\n LABELS = {{ $labels }}"
```

Listing 6.1: Beispiel für eine Prometheus Alert Regel (Berthe 2021).

Grundlegende Alerting Regeln wurden dem *Awesome Prometheus Alerts* GitHub Repository[4] von Samuel Berthe entnommen (Berthe 2021). Das Repository stellt eine Sammlung nützlicher von der GitHub Community zusammengetragenen Alerting Regeln zur Verfügung und ist zum Zeitpunkt der Recherche mit über 2800 Stars sehr beliebt.

Folgende Regeln werden verwendet:

1. ConsulServiceHealthcheckFailed
2. ConsulAgentUnhealthy
3. HostOutOfMemory
4. HostMemoryUnderMemoryPressure
5. HostOutOfDiskSpace
6. HostDiskWillFillIn24Hours
7. HostOutOfInodes
8. HostInodesWillFillIn24Hours
9. HostUnusualDiskReadLatency
10. HostUnusualDiskWriteLatency
11. HostHighCpuLoad
12. HostCpuStealNoisyNeighbor
13. HostSwapIsFillingUp
14. HostSystemdServiceCrashed
15. HostOomKillDetected

[4] https://github.com/samber/awesome-prometheus-alerts

16. HostNetworkReceiveErrors
17. HostNetworkTransmitErrors
18. HostNetworkInterfaceSaturated
19. PrometheusTargetMissing
20. PrometheusAllTargetsMissing
21. PrometheusTooManyRestarts
22. PrometheusAlertmanagerConfigurationReloadFailure
23. PrometheusAlertmanagerConfigNotSynced
24. PrometheusNotConnectedToAlertmanager
25. PrometheusRuleEvaluationFailures
26. PrometheusRuleEvaluationSlow
27. PrometheusAlertmanagerNotificationFailing
28. PrometheusTargetScrapingSlow
29. TraefikServiceDown
30. TraefikHighHttp4xxErrorRateService
31. TraefikHighHttp5xxErrorRateService

Alle Regeln mit dem Prefix *Consul* betreffen die Metriken, welche vom Consul Exporter erhoben werden. So wird zum Beispiel ein Alert versendet, sobald der Health Check eines registrierten Services fehlschlägt.

Regeln mit dem *Host* Prefix betreffen die Metriken, welche von den Node Exportern gesammelt werden. Besonders hilfreich sind hier die Regeln *HostOutOfMemory* und *HostOutOfDiskSpace*, welche dabei helfen, die häufigsten Probleme, zu wenig Arbeitsspeicher oder zu wenig Platz auf der Festplatte, zu erkennen.

Die Regeln mit *Prometheus* Prefix betreffen allgemeine Ereignisse innerhalb der Prometheus und Alertmanager Instanzen. So wird beispielsweise durch *Prometheus-TargetMissing* ein Alert ausgelöst, sobald ein Target nicht mehr erreichbar ist.

Abschließend noch die Regeln mit *Traefik* Prefix zur Erkennung ungewöhnlich vieler HTTP Fehler oder eines nicht funktionierenden Services innerhalb eines Traefik Proxies.

6.3 Definition grundlegender Dashboards

Um die Metriken, welche bereits erhoben werden, mithilfe von Dashboards darstellen zu können, wurden folgende von der Grafana Community erstellte Dashboards[5] als Basis verwendet. Die Dashboards wurden nach der Anzahl der Downloads und der Kompatibilität mit der verwendeten Grafana Version 8.0.6 ausgewählt.

Das Alertmanager Dashboard (Abbildung 6.6) zeigt Informationen zum Alertmanager Cluster, darunter zum Beispiel wie viele Alerts momentan aktiv sind, wie viele Alertmanager sich im Cluster befinden und welcher Alertmanager wie viele Benachrichtigungen versand hat.

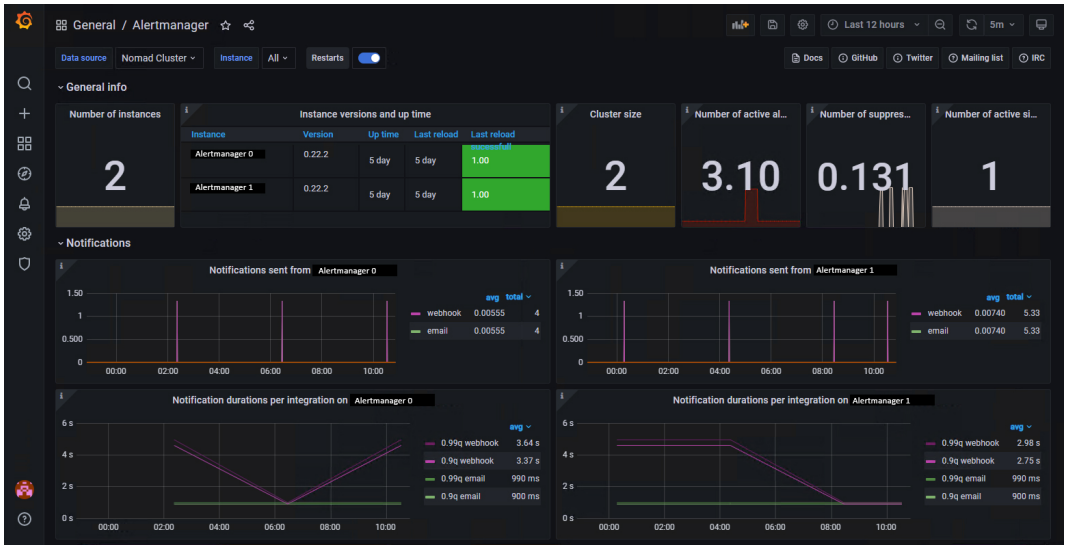

Abbildung 6.6: Alertmanager Dashboard von Martin Chodur (https://grafana.com/grafana/dashboards/9578).

[5] https://grafana.com/grafana/dashboards

Das Consul Exporter Dashboard (Abbildung 6.7) zeigt Informationen zum Consul Cluster. Zum Beispiel wie viele Services aktiv sind, wie viele Clients sich im Cluster befinden und bei wie vielen Services der Health Check fehlschlägt.

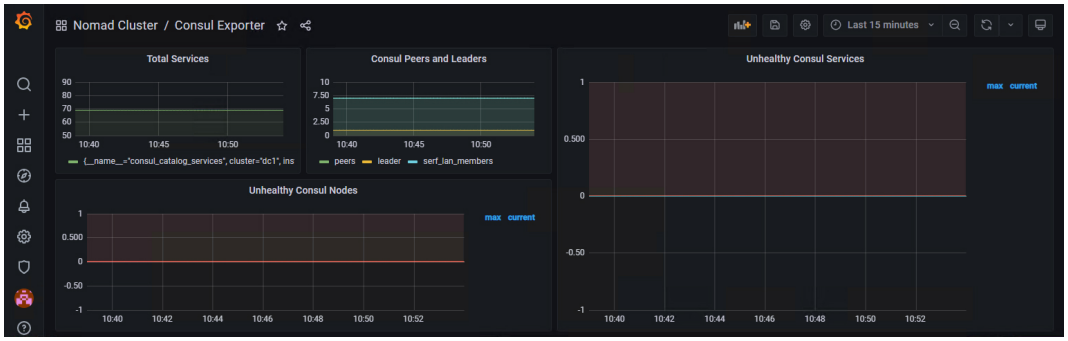

Abbildung 6.7: Consul Exporter Dashboard von Joey Yang (https://grafana.com/grafana/dashboards/12049).

Das Node Exporter Dashboard (Abbildung 6.8) liefert sehr detaillierte Informationen über die überwachten Systeme, darunter zum Beispiel die Netzwerkaktivität sowie die Nutzung der verfügbaren Festplatten. Es sind jedoch auch speziellere Metriken wie die Synchronisierung der Systemuhr oder die aktuell verwendete Anzahl von Inodes (Verweise des Betriebssystems auf Dateien und Ordner im Dateisystem) dargestellt. Diese detaillierteren Metriken wurden aus Platzgründen nicht in den Screenshot aufgenommen.

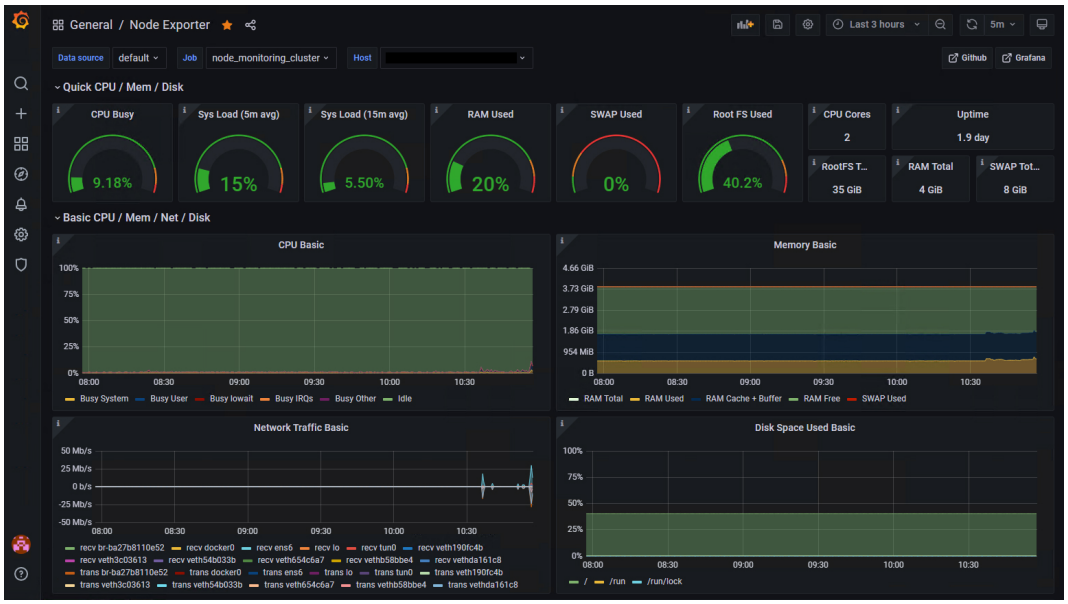

Abbildung 6.8: Node Exporter Dashboard von Ricardo Fraile (https://grafana.com/grafana/dashboards/1860).

Das Traefik Dashboard (Abbildung 6.9) zeigt grundlegende Informationen zu den verschiedenen registrierten Services innerhalb der Traefik Proxies. Darunter befinden sich zum Beispiel die Anzahl der Anfragen und HTTP Statuscodes sowie die durchschnittliche Antwortzeit.

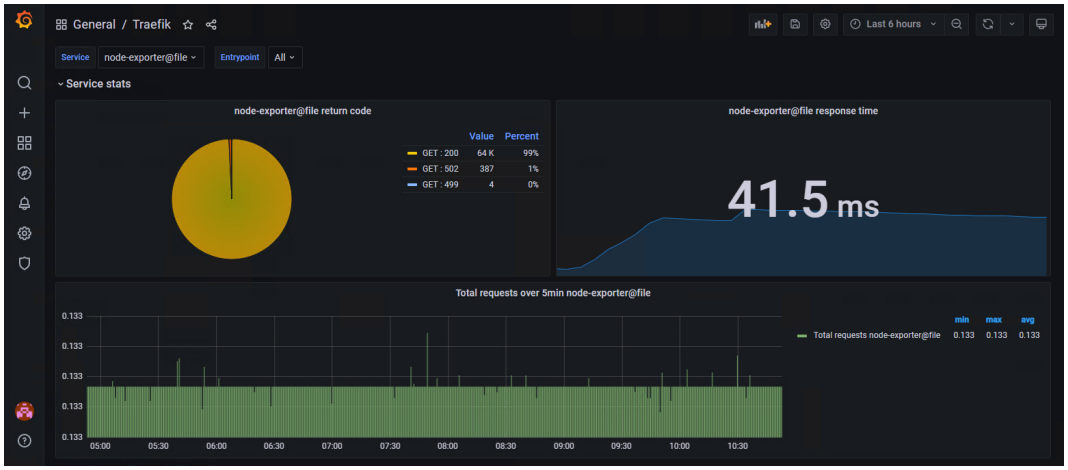

Abbildung 6.9: Traefik Dashboard von Thomas Cheronneau (https://grafana.com/grafana/dashboards/4475).

Die Dashboards sind noch umfangreicher als auf den Screenshots dargestellt. Das Node Exporter Dashboard hat zum Beispiel 16 Gruppierungen, welche jeweils mehr oder weniger genauso viele Elemente enthalten wie auf Abbildung 6.8 zu sehen. Daher werden hier jeweils nur die ersten ein bis zwei Gruppierungen gezeigt.

Kapitel 7

Evaluation der Umsetzung

7.1 Funktionale Anforderungen

Wie in Tabelle 7.1 zu sehen, sind alle funktionalen Anforderungen bis auf FA5 und FA10 vollständig erfüllt.

Anforderung	Umsetzungsgrad
FA1: Monitoring von Hosts und Containern	●
FA2: Monitoring von Kundensystemen	●
FA3: Monitoring von Systemen vor Ort	●
FA4: Authentifizierung für die Benutzeroberflächen	●
FA5: Abbildung bestehender Rechtestrukturen	◑
FA6: Benachrichtigungen über Microsoft Teams	●
FA7: Benachrichtigungen über E-Mail	●
FA8: Erstellung verschiedener Dashboards	●
FA9: Gruppierungen innerhalb der Dashboards	●
FA10: „Ampel“ als Indikator für Systemzustand	◑
FA11: Flexible Aufbewahrungsrichtlinien	●

Tabelle 7.1: Evaluation der Umsetzung der funktionalen Anforderungen.

Die Anforderungen FA1 bis FA3 beziehen sich auf das Monitoring der Systeme beziehungsweise die Fähigkeit der Monitoring Lösung, verschiedene Arten von Systemen anzubinden. Da Prometheus nicht nur Anwendungen überwachen kann, welche eine native Metriken Schnittstelle implementieren, sondern ebenfalls die Überwachung von Systemen mithilfe der verschiedenen Exporter ermöglicht, sind diese Anforderungen erfüllt. Die einzige Bedingung für die Überwachung ist daher das Vorhandensein einer Netzwerkverbindung zwischen den Zielsystemen und den Prometheus Instanzen.

FA5 und FA6 betreffen die Authentifizierung der Benutzeroberflächen sowie die Abbildung von bestehenden Rechtestrukturen. Da Organizr sowie Grafana jeweils an einen LDAP angebunden werden können, ist die Authentifizierung für die Benutzeroberflächen mithilfe der zentralen Benutzerkonten möglich. Die Abbildung von Rechtestrukturen in Bezug auf die Authentifizierung verschiedener Bereiche der Benutzeroberflächen ist ebenfalls möglich. Lediglich die Abbildung der Rechtestrukturen für die Benachrichtigungen der Alertmanager lassen sich nicht über den LDAP verwirklichen. Dies liegt daran, dass die Konfiguration für die Empfänger von Benachrichtigungen nur innerhalb der Konfigurationsdatei der Alertmanager eingestellt werden kann, somit müssen die Rechtestrukturen hier manuell abgebildet werden.

FA6 und FA7 beziehen sich auf den Versand von Benachrichtigungen über Microsoft Teams sowie E-Mail. Mithilfe des Prom2Teams Proxy Containers können über die generische Webhook Versandart der Alertmanager Benachrichtigungen an Microsoft Teams gesendet werden. E-Mail-Benachrichtigungen werden bereits nativ unterstützt.

Die Erstellung verschiedener Dashboards sowie das Erstellen von Gruppierungen innerhalb der Dashboards wird von Grafana nativ unterstützt. Dashboards können außerdem für mehrere Targets verwendet werden. So bietet zum Beispiel das Node Exporter Dashboard die Möglichkeit zur Auswahl eines Hosts aus einem bestimmten Prometheus Job. Somit sind FA8 und FA9 vollständig umgesetzt.

FA10 wird nur als teilweise umgesetzt eingestuft, da zum Zeitpunkt der Implementierung der Lösung noch nicht genug Daten und Erfahrungswerte vorliegen, um einen solchen Indikator umzusetzen. Prinzipiell können jedoch verschiedene Elemente innerhalb eines Dashboards hierfür verwendet werden, da hinter jeder Darstellung das Ergebnis einer PromQL Abfrage steht. So könnte zum Beispiel ein Wert anhand verschiedener kritischer Parameter berechnet und in der Darstellung verschiedene Grenzwerte definiert werden, anhand derer ersichtlich wird, ob der Wert sich im kritischen Bereich befindet oder nicht.

Da innerhalb des IONOS S3 Aufbewahrungsrichtlinien für die verschiedenen Buckets definiert werden können, wird diese Anforderung erfüllt.

7.2 Qualitätsanforderungen

Wie in Tabelle 7.2 zu sehen, sind alle Qualitätsanforderungen bis auf QA4 und QA5 vollständig erfüllt.

Anforderung	Umsetzungsgrad
QA1: Verteiltes System	●
QA2: Skalierbarkeit des Systems	●
QA3: Ein Prometheus HA Paar pro Cluster oder Einheit	●
QA4: ~99% Verfügbarkeit	◑
QA5: Verschlüsselte Kommunikation der Komponenten	◑
QA6: Verschlüsselter S3 Speicher	●
QA7: Lückenloses einsammeln von Metriken	●
QA8: Zeitnahe Benachrichtigung über Vorfälle	●
QA9: Übersichtliche Dashboards	●

Tabelle 7.2: Evaluation der Umsetzung der Qualitätsanforderungen.

Die Anforderungen QA1 bis QA4 betreffen die Architektur beziehungsweise Verfügbarkeit der Monitoring Lösung. QA1 bis QA3 werden vollständig erfüllt da die Monitoring Lösung als verteiltes System entworfen, die Skalierbarkeit durch Thanos gegeben und ein Prometheus HA Paar im Nomad Cluster betrieben wird. Anhand des Systementwurfs ist eine hohe Verfügbarkeit zu erwarten. Da dies jedoch erst nach einem längeren Zeitraum bestätigt werden kann, ist QA4 zum Zeitpunkt der Evaluation nur teilweise erfüllt.

QA5 und QA6 betreffen die Verschlüsselung der Kommunikation und der erhobenen Metriken. Die verschlüsselte Kommunikation zwischen den Komponenten ist zwar möglich, zum Zeitpunkt der Umsetzung ist dies jedoch zumindest bei Prometheus, Alertmanager und Thanos noch eine experimentelle Funktion. Deswegen wird die Kommunikation innerhalb des privaten Netzes nicht verschlüsselt. Eine Abfrage von Metriken über HTTPS ist jedoch möglich. Die Daten im IONOS S3 werden mit serverseitiger Verschlüsselung gespeichert, daher ist diese Anforderung erfüllt.

QA6 und QA7 werden erfüllt, da die Prometheus Instanzen sowie die Alertmanager Instanzen in einem HA Setup betrieben werden. Daher ist die Ausfallsicherheit auf Anwendungsebene gegeben und Metriken können lückenlos gesammelt werden. Da die Auswertung der Alerting Regeln der Prometheus Instanzen lokal erfolgt und die Kommunikation mit den Alertmanager Instanzen im privaten Netz stattfindet, bleibt als einziger SPOF die Verbindung zum Internet. Hierauf kann jedoch kein

Einfluss genommen werden, da die gesamte Monitoring Lösung mithilfe des IaaS Angebots von IONOS betrieben wird und somit die Verbindung zum Internet in der Verantwortung von IONOS liegt.

Aufgrund der vielfältigen Möglichkeiten zur Gestaltung von Dashboards innerhalb von Grafana und der Erfüllung der funktionalen Anforderungen in Bezug auf die Visualisierung der gesammelten Metriken, ist QA9 erfüllt.

Kapitel 8

Zusammenfassung und Ausblick

Im Verlauf dieser Arbeit wurde eine funktionsfähige Monitoring Lösung für die Cloud der bitExpert AG konzeptioniert und in die bestehende Infrastruktur integriert. Der Großteil der ermittelten Anforderungen konnte erfüllt werden, und erste Systeme wurden an das Monitoring angebunden.

Im Rahmen der Anforderungsanalyse wurden die Anforderungen der bitExpert AG an die Monitoring Lösung ermittelt und Randbedingungen abgeleitet. Außerdem wurden potenzielle Betriebsmöglichkeiten ermittelt und evaluiert, um die bestmögliche Lösung für die Integration in die bestehende Infrastruktur unter Berücksichtigung von Ausfallsicherheit und Sicherheit zu finden. Schließlich schied das Managed Kubernets Angebot von IONOS für den Betrieb des Monitoring Frontends aufgrund mangelnder Flexibilität aus, und es wurde ein klassisches VM Setup in einer anderen Rechenzentrumsregion als der des Nomad Cluster gewählt.

Während des Systementwurfs wurden aktuelle Cloud Technologien wie das Thanos Projekt verwendet, um die bewährte Monitoring Lösung Prometheus nicht nur ausfallsicher, sondern auch skalierbar in die Infrastruktur zu integrieren. Die Komponenten für Monitoring und Alerting, Prometheus und Alertmanager, wurden als HA Setup in das Nomad Cluster integriert. Die VM für das Monitoring Frontend wurde über OpenVPN mit dem Nomad Cluster verbunden, um die teilweise unverschlüsselte Kommunikation der Komponenten abzusichern und die Abfrage von Metriken von den Prometheus Instanzen von einer zentralen Stelle aus zu ermöglichen. Auf diese Art und Weise ist es in Zukunft möglich, noch weitere Cluster an das Monitoring Frontend anzubinden und gleichzeitig das Monitoring und das Alerting von der Darstellung der Daten zu trennen. Durch diese Trennung können im Fall eines Systemausfalls in einem Cluster immer noch Metriken aus anderen

Clustern abgefragt und gleichzeitig auf die bis zum Zeitpunkt des Ausfalls im IONOS S3 gespeicherten Metriken zugegriffen werden.

Die Umsetzung sowie das Konfigurationsmanagement der konzipierten Lösung wurde unter Beachtung der IaC Prinzipien mithilfe von Terraform, Ansible und CI/CD Pipelines vollständig automatisiert. Somit liegt in gewissem Maße bereits eine Dokumentation der Infrastruktur vor, und es ist bei Änderungen mit einem Knopfdruck möglich, diese umzusetzen oder auf eine vorherige Version zurückzuspringen.

Die Evaluation der Umsetzung zeigte schließlich auf, wo noch Verbesserungspotenzial besteht. Teils wurde dies durch noch experimentelle Funktionen der verwendeten Komponenten oder schlicht durch die mangelnde Zeit für eine langfristige Beobachtung des umgesetzten Entwurfs bedingt.

Mit Blick in die Zukunft und bei weiterem Wachstum der Infrastruktur, sollten die Komponenten des Monitoring Frontends ebenfalls ausfallsicher betrieben werden. Außerdem sollte die Entwicklung insbesondere von Prometheus, Alertmanager und Thanos weiter beobachtet werden, um zukünftige Sicherheitsfunktionen wie die Authentifizierung der API-Schnittstellen so bald wie möglich verwenden zu können. Dies wäre ein Weg, den SPOF, den die VPN-Verbindung zum Monitoring Frontend darstellt, zu eliminieren.

Abschließend bleibt zu sagen, dass die Monitoring Lösung im Rahmen der gegebenen Randbedingungen so flexibel und ausfallsicher wie möglich konzipiert wurde, jedoch immer Optimierungspotenzial besteht, da sich die verwendeten Technologien stetig weiterentwickeln.

Abkürzungsverzeichnis

API	Application Programming Interface
AWS	Amazon Web Services
CD	Continuous Delivery
CI	Continuous Integration
DCD	Data Center Designer
DNS	Domain Name System
gRPC	Google Remote Procedure Calls
HA	High Availability
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IaaS	Infrastructure as a Service
IaC	Infrastructure as Code
KISS	Keep it simple, stupid
LAN	Local Area Network
LDAP	Lightweight Directory Access Protocol
PromQL	Prometheus Query Language
RAID	Redundant Array of Inexpensive Disks
REST	Representational State Transfer
RPC	Remote Procedure Calls
S3	Simple Storage Service
SLA	Service Level Agreement
SPOF	Single Point of Failure
SSH	Secure Shell
TLS	Transport Layer Security
TSDB	Time Series Database
VM	Virtuelle Maschine
VPN	Virtual Private Network
WAN	Wide Area Network
WinRM	Windows Remote Management
YAML	YAML Ain’t Markup Language

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Literatur

Ahluwalia, Kanwardeep Singh und Atul Jain (2006). „High Availability Design Patterns“. In: *Proceedings of the 2006 Conference on Pattern Languages of Programs*. PLoP '06. Portland, Oregon, USA: Association for Computing Machinery. DOI: 10.1145/1415472.1415494. URL: https://doi.org/10.1145/1415472.1415494.

Amazon (2021a). *Amazon S3 Funktionen*. URL: https://aws.amazon.com/de/s3/features/?nc=sn&loc=2 (besucht am 23. 06. 2021).

— (2021b). *Amazon S3 Übersicht*. URL: https://aws.amazon.com/de/s3/?nc=sn&loc=1 (besucht am 23. 06. 2021).

Ansible (2021). *Validating tasks: check mode and diff mode*. URL: https://docs.ansible.com/ansible/latest/user_guide/playbooks_checkmode.html (besucht am 27. 07. 2021).

Bairavasundaram, Lakshmi N. u. a. (Nov. 2008). „An Analysis of Data Corruption in the Storage Stack“. In: *ACM Trans. Storage* 4.3. DOI: 10.1145/1416944.1416947.

Berthe, Samuel (2021). *Awesome Prometheus Alerts*. URL: https://awesome-prometheus-alerts.grep.to/ (besucht am 04. 08. 2021).

Birman, Ken (Okt. 2007). „The Promise, and Limitations, of Gossip Protocols“. In: *SIGOPS Oper. Syst. Rev.* 41.5, S. 8–13. DOI: 10.1145/1317379.1317382.

CNCF (2020). *Cloud Native Computing Foundation Survey 2020*. URL: https://www.cncf.io/wp-content/uploads/2020/11/CNCF_Survey_Report_2020.pdf (besucht am 23. 07. 2021).

Containous, Traefik Labs (2021a). *BasicAuth*. URL: https://doc.traefik.io/traefik/middlewares/basicauth/ (besucht am 02. 08. 2021).

Containous, Traefik Labs (2021b). *ForwardAuth*. URL: https://doc.traefik.io/traefik/middlewares/forwardauth/ (besucht am 08. 07. 2021).

— (2021c). *Prometheus*. URL: https://doc.traefik.io/traefik/observability/metrics/prometheus/ (besucht am 02. 08. 2021).

— (2021d). *Service Mesh in Traefik Enterprise*. URL: https://doc.traefik.io/traefik-enterprise/operations/service-mesh/ (besucht am 22. 07. 2021).

— (2021e). *Traefik EE Features*. URL: https://doc.traefik.io/traefik-enterprise/v2.3/features/ (besucht am 29. 06. 2021).

— (2021f). *Welcome to Traefik Enterprise!* URL: https://doc.traefik.io/traefik-enterprise/v2.3/ (besucht am 29. 06. 2021).

Das, A., I. Gupta und A. Motivala (2002). „SWIM: scalable weakly-consistent infection-style process group membership protocol“. In: *Proceedings International Conference on Dependable Systems and Networks*, S. 303–312. DOI: 10.1109/DSN.2002.1028914.

Dix, Paul (2021). *Why Time Series Matters for Metrics, Real-Time Analytics and Sensor Data*. URL: http://get.influxdata.com/rs/972-GDU-533/images/why%20time%20series.pdf (besucht am 21. 07. 2021).

Drake, Sam u. a. (2005). „Architecture of Highly Available Databases“. In: *Service Availability*. Hrsg. von Miroslaw Malek, Manfred Reitenspieß und Jörg Kaiser. Berlin, Heidelberg: Springer Berlin Heidelberg, S. 1–16.

Ebert, Christof (2019). *Systematisches Requirements Engineering*. ger. 6. Aufl. dpunkt.verlag. URL: https://content-select.com/de/portal/media/view/5d76655d-c984-4260-81a0-7e37b0dd2d03.

Experte A, Sascha Nockel (Juni 2021). *Experteninterview mit Experte A*.

Experte B, Sascha Nockel (Juni 2021). *Experteninterview mit Experte B*.

GitLab (2021a). *All GitLab Features*. URL: https://about.gitlab.com/features/ (besucht am 27. 07. 2021).

— (2021b). *GitLab managed Terraform State*. URL: https://docs.gitlab.com/ee/user/infrastructure/terraform_state.html (besucht am 24. 07. 2021).

— (2021c). *GitLab Runner*. URL: https://docs.gitlab.com/runner/ (besucht am 27. 07. 2021).

Grafana (2021a). *LDAP Authentication*. URL: https://grafana.com/docs/grafana/latest/auth/ldap/ (besucht am 29. 08. 2021).

— (2021b). *Manage users*. URL: https://grafana.com/docs/grafana/latest/manage-users/ (besucht am 21. 07. 2021).

— (2021c). *Overview*. URL: https://grafana.com/grafana/ (besucht am 09. 06. 2021).

HalianElf (2019). *LDAP Authentication*. URL: https://docs.organizr.app/books/setup-features/page/ldap-authentication (besucht am 08. 07. 2021).

— (2021). *ServerAuth*. URL: https://docs.organizr.app/books/setup-features/page/serverauth (besucht am 08. 07. 2021).

HashiCorp (2018). *Introduction to HashiCorp Terraform with Armon Dadgar*. URL: https://www.youtube.com/watch?v=h970ZBgKINg (besucht am 24. 07. 2021).

— (2021a). *Automated Service Networking with Consul*. URL: https://www.nomadproject.io/use-cases/automated-service-networking-with-consul (besucht am 07. 06. 2021).

— (2021b). *Basic Terraform CLI Features*. URL: https://www.terraform.io/docs/cli/commands/index.html (besucht am 24. 07. 2021).

— (2021c). *Consul Architecture*. URL: https://www.consul.io/docs/architecture (besucht am 29. 06. 2021).

— (2021d). *Consul Connect*. URL: https://www.consul.io/docs/connect (besucht am 22. 07. 2021).

— (2021e). *Introduction to Consul*. URL: https://www.consul.io/docs/intro (besucht am 29. 06. 2021).

— (2021f). *Introduction to Vault*. URL: https://www.vaultproject.io/docs/what-is-vault (besucht am 29. 06. 2021).

HashiCorp (2021g). *Nomad Architecture*. URL: https://www.nomadproject.io/docs/internals/architecture (besucht am 09. 06. 2021).

— (2021h). *Nomad vs. Kubernetes*. URL: https://www.nomadproject.io/docs/nomad-vs-kubernetes (besucht am 07. 06. 2021).

— (2021i). *Non-Containerized Application Orchestration*. URL: https://www.nomadproject.io/use-cases/non-containerized-application-orchestration (besucht am 07. 06. 2021).

— (2021j). *Simple Container Orchestration*. URL: https://www.nomadproject.io/use-cases/simple-container-orchestration (besucht am 07. 06. 2021).

— (2021k). *Terraform vs. Chef, Puppet, etc.* URL: https://www.terraform.io/intro/vs/chef-puppet.html (besucht am 09. 08. 2021).

— (2021l). *The Two Million ContainerChallenge*. URL: https://www.hashicorp.com/c2m (besucht am 07. 06. 2021).

— (2021m). *Understand Consul Service Mesh*. URL: https://learn.hashicorp.com/tutorials/consul/service-mesh?in=consul/gs-consul-service-mesh (besucht am 09. 08. 2021).

— (2021n). *Vault Architecture*. URL: https://www.vaultproject.io/docs/internals/architecture (besucht am 29. 06. 2021).

— (2021o). *Vault High Availability Mode*. URL: https://www.vaultproject.io/docs/concepts/ha (besucht am 22. 07. 2021).

Humble, Jez und David Farley (2010). *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. 1st. Addison-Wesley Professional.

IONOS (2021a). *IaaS: Hochgradig skalierbare IT-Infrastruktur aus der Cloud*. URL: https://www.ionos.de/digitalguide/server/knowhow/iaas-infrastructure-as-a-service/ (besucht am 04. 06. 2021).

— (2021b). *S3 Object Storage*. URL: https://cloud.ionos.de/storage/object-storage (besucht am 23. 06. 2021).

ISG (2019). *ISG ProviderLens Quadrant Report*. URL: https://ce2.uicdn.net/shopsfar/pdf/IONOS-ISG-Quadrant-Report-Public-Cloud-Germany-2019.pdf (besucht am 04. 06. 2021).

Lefevre, Kevin (2021). *Multi-cluster monitoring with Thanos*. URL: https://www.cncf.io/blog/2021/03/15/multi-cluster-monitoring-with-thanos/ (besucht am 23. 07. 2021).

Liu, Duo u. a. (2020). „Downsizing Without Downgrading: Approximated Dynamic Time Warping on Nonvolatile Memories“. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39.1, S. 131–144. DOI: 10.1109/TCAD.2018.2878182.

Lucas Serven, Dominic Green (2019). *Intro to Thanos: Scale Your Prometheus Monitoring With Ease*. URL: https://static.sched.com/hosted_files/kccncna19/e4/CNCFSanDiego-IntroductionToThanos.pdf (besucht am 09. 06. 2021).

Mesbahi, Mohammad Reza, Amir Masoud Rahmani und Mehdi Hosseinzadeh (Juli 2018). „Reliability and high availability in cloud computing environments: a reference roadmap“. In: *Human-centric Computing and Information Sciences* 8.1, S. 20. DOI: 10.1186/s13673-018-0143-8. URL: https://doi.org/10.1186/s13673-018-0143-8.

Morris, Kief (2016). *Infrastructure as Code: Managing Servers in the Cloud*. 1st. O'Reilly Media, Inc.

Ongaro, Diego und John Ousterhout (Juni 2014). „In Search of an Understandable Consensus Algorithm“. In: *2014 USENIX Annual Technical Conference (USENIX ATC 14)*. Philadelphia, PA: USENIX Association, S. 305–319. URL: https://www.usenix.org/system/files/conference/atc14/atc14-paper-ongaro.pdf.

Prometheus (2021a). *Alerting Rules*. URL: https://prometheus.io/docs/prometheus/latest/configuration/alerting_rules/ (besucht am 08. 06. 2021).

— (2021b). *Alertmanager*. URL: https://prometheus.io/docs/alerting/latest/alertmanager/ (besucht am 09. 06. 2021).

— (2021c). *Client Libraries*. URL: https://prometheus.io/docs/instrumenting/clientlibs/ (besucht am 08. 06. 2021).

— (2021d). *Data Model*. URL: https://prometheus.io/docs/concepts/data_model/ (besucht am 08. 06. 2021).

Prometheus (2021e). *Exporters and Integrations*. URL: https://prometheus.io/docs/instrumenting/exporters/ (besucht am 08. 06. 2021).

— (2021f). *Federation*. URL: https://prometheus.io/docs/prometheus/latest/federation/ (besucht am 09. 08. 2021).

— (2021g). *Frequently Asked Questions*. URL: https://prometheus.io/docs/introduction/faq/#can-prometheus-be-made-highly-available (besucht am 09. 08. 2021).

— (2021h). *Metric Types*. URL: https://prometheus.io/docs/concepts/metric_types/ (besucht am 08. 06. 2021).

— (2021i). *Overview*. URL: https://prometheus.io/docs/introduction/overview/ (besucht am 08. 06. 2021).

— (2021j). *Security Model*. URL: https://prometheus.io/docs/operating/security/ (besucht am 07. 07. 2021).

— (2021k). *Storage*. URL: https://prometheus.io/docs/prometheus/latest/storage/ (besucht am 08. 06. 2021).

RedHat (2017). *Ansible In Depth Whitepaper*. URL: https://www.ansible.com/hubfs/pdfs/Ansible-InDepth-WhitePaper.pdf (besucht am 26. 07. 2021).

— (2021). *Das Service Mesh – Funktionsweise und Vorteile*. URL: https://www.redhat.com/de/topics/microservices/what-is-a-service-mesh (besucht am 22. 07. 2021).

Serrano, Nicolás, Gorka Gallardo und Josune Hernantes (2015). „Infrastructure as a Service and Cloud Technologies“. In: *IEEE Software* 32.2, S. 30–36. DOI: 10.1109/MS.2015.43.

Tak, Byung Chul, Bhuvan Urgaonkar und Anand Sivasubramaniam (2011). „To Move or Not to Move: The Economics of Cloud Computing“. In: *Proceedings of the 3rd USENIX Conference on Hot Topics in Cloud Computing*. HotCloud'11. Portland, OR: USENIX Association, S. 5. DOI: 10.5555/2170444.2170449.

Thanos (2021a). *Quick Tutorial*. URL: https://thanos.io/tip/thanos/quick-tutorial.md/ (besucht am 08. 06. 2021).

— (2021b). *Security Policy*. URL: https://thanos.io/tip/thanos/security.md/ (besucht am 07. 07. 2021).

Vishwanath, Kashi Venkatesh und Nachiappan Nagappan (2010). „Characterizing Cloud Computing Hardware Reliability“. In: *Proceedings of the 1st ACM Symposium on Cloud Computing*. SoCC ’10. Indianapolis, Indiana, USA: Association for Computing Machinery, S. 193–204. DOI: 10.1145/1807128.1807161.