Einrichtung einer verteilten Monitoringlösung auf Basis von Prometheus und M3 im Bereich des behördlichen Gesundheitswesens

Projektbericht

von Nico Kahlert

26.02.2021

Ausbildungsberuf:	Fachinformatiker für Systemintegration
Durchführungszeitraum:	26.02.2021 - 28.04.2021
Verfasser / Prüfling:	Nico Kahlert
Prüfung:	Sommer 2021
Identnummer:	594872
Prüflingsnummer:	111 3188
Ausbildungsbetrieb:	Netzlink Informationstechnik GmbH Westbahnhof 11 38118 Braunschweig

Vorwort

Um in diesem Bericht von der Wiedergabe von Rollenklischees und Stereotypen abzusehen, wird von der Nutzung des generischen Maskulinum abgesehen und stattdessen ein sogenanntes Gendersternchen zur Inklusion aller Geschlechtsidentitäten verwendet. Weiterhin sind komplett großgeschriebene Wörter im Glossar erläutert. Bedeutungsvolle Begriffe sind kursiv geschrieben. Grafische Schaubilder und Diagramme befinden sich separiert im Anhang und werden im Text mittels Verweis gekennzeichnet. Die in dem Bericht wiedergegebenen Projektphasen werden mit horizontalen Linien gekennzeichnet. Die Farben dieser Linien entsprechen denen der Tabellen (Seite 26).

Inhaltsverzeichnis

1 Einleitung

Im Zuge der andauernden CORONAKRISE wurde der Auftrag zum Betrieb der Pandemiemanagementsoftware ”Surveillance Outbreak Response Management and Analysis System” (SORMAS) in den Rechenzentren des ITZBund an die Netzlink Informationstechnik GmbH (Netzlink) erteilt. Aufgrund eines raschen Anstiegs von zu überwachenden Maschinen in unserer Infrastruktur im Rechenzentrum des ITZBUND stellte sich heraus, dass die derzeitige Monitoringlösung der Aufgabe nicht mehr standhalten kann. Diese Infrastruktur ist für die alleinige Überwachung der dortigen Systeme verantwortlich, welche eine *Software as a Service (SaaS)* des behördlichen Gesundheitswesen bereitstellen und als eine sogenannte *kritische Infrastruktur* eingestuft werden. Zur Verbesserung muss ein passendes Monitoringkonzept erarbeitet werden. Dieser Entwicklungsbedarf und die Umsetzung des Projektes wird in dieser Dokumentation ganzheitlich beschrieben.

1.1 Vorstellung des Kunden

Das Bundesministerium für Gesundheit (BMG) ist vor allem zuständig für die Erarbeitung von Gesetzentwürfen für das deutsche Gesundheitssystem und wird von der*m Bundesminister*in für Gesundheit geleitet. Zur Zeit ist dies Jens Spahn von der Christlich Demokratische Union (CDU). Dem BMG, welchem jährlich ungefähr 15 Milliarden Euro Haushalt zur Verfügung steht, gehören circa 700 Bedienstete an. Es ist die höchste Instanz im Gesundheitssystem und unter anderem als Akteur wegweisend in der Krankheitsprävention [1].

1.2 Projekteinordnung

Da es sich beim Betrieb von SORMAS für die behördlichen Gesundheitsämter um eine kritische Infrastruktur handelt, muss zu jedem Zeitpunkt sichergestellt werden, dass der Dienst voll funktionsfähig ist. Die Basis dafür ist ein Monitoringsystem, welches die Anforderungen an Stabilität und Strapazierfähigkeit unterstützt und so eine dauerhafte Serviceüberwachung möglich macht. Im Leitfaden für IT-Service-Management ITIL, nach welchem der Service von Netzlink strukturiert ist, ist das Monitoring eine wichtige Datenquelle für das *Availability Management* und somit ein Indikator für die Zufriedenheit des Kunden [27].

1.3 Auswahl des Projektes

Die meiste Zeit verbrachte ich im OpenSolutions-Team, welches für den Betrieb und Architektur von OpenSource-Software bei Kund*innen zuständig ist. Aus diesem Grund wollte ich auch das Abschlussprojekt in diesem Bereich durchführen.

1.4 Verwendete Software

Im Folgenden werden die Anwendungsbereiche der in der Umsetzung verwendeten Softwareprodukte beschrieben.

1.4.1 Sormas

SORMAS ist eine Software zum Pandemiemanagement und wurde vom Helmholtz Zentrum für Infektionsforschung (HZI) aus Braunschweig zusammen mit der Firma Vitagroup für die Ebola-Epidemie von Ostafrika 2014 entwickelt. In Folge der Coronakrise wird diese Webanwendung für den behördlichen Gesundheitsdienst im Auftrag des BMG von Netzlink im Rechenzentrum des ITZBund betrieben. Gesundheitsamtsmitarbeiter*innen nutzen sie zur Nachverfolgung von Kontakten und Quarantänezeiträumen. Die Software ist unter der Apache OpenSource-Lizenz geschützt und kann von jeder*m frei betrieben werden. Geschrieben ist die Applikation in Java und speichert ihre Daten in einer relationalen Postgresdatenbank. Die Software wird in Produktion bei Netzlink in Docker-Containern betrieben, um ein möglichst leichtes Lifecyclemanagement und problemlose Day2Ops zu garantieren. Aktuell wird je Gesundheitsamt eine SORMAS-Instanz inklusive Datenbank und Proxy auf einer virtuellen Maschine ausgerollt. [2]

1.4.2 Prometheus

Prometheus ist ein freies OpenSource Programm, welches ursprünglich bei dem in den USA ansässigen Streamingunternehmen Soundcloud entwickelt wurde. Es dient dem Sammeln, Aggregieren, Speichern und Abfragen von Betriebsdaten von Sofwaresystemen. Als Grundlage für das Sammeln der Daten dient das prometheuseigene, auf der extended Backus-Naur Form (EBNF) basierte Datenformat in Textform (siehe Abbildung 8 auf Seite 20). In einem typischen Whiteboxmonitoringszenario werden diese Daten, welche auch als Metriken bezeichnet werden, über einen Hypertext Transfer Protocol (HTTP)-Endpunkt mit dem Pfad */metrics* bereitgestellt. Diese werden in regelmäßigen, einstellbaren Abständen von dem Prometheusserver abgegriffen (scraping) und auf dessen lokalen Filesystem persistiert. Es ist auch möglich, externe Timeseriesdatenbanken mittels des *Prometheusremoteprotocols* zu nutzen. Die Metriken können über die Representational State Transfer (REST) API oder der Weboberfläche abgegriffen werden. Zum Einholen der Metriken wird die Domain Specific Language (DSL) Prometheus Query Language (PromQL) verwendet [19] (siehe Abbildung 9 auf Seite 20). Innerhalb von containerbasierten Microserviceinfrastrukturen gilt Prometheus im Verbund mit Grafana als Standard.

1.4.3 Nodexporter

Der Nodexporter ist ein in Golang entwickelter Metricsexporter zum Sammeln von Betriebsdaten des darunterliegenden Betriebsystems. Das Programm zählt zu den Standardwerkzeugen der Monitoringlösung um Prometheus [20].

1.4.4 Grafana

Grafana ist ein universelles, webbasiertes Werkzeug zur grafischen Darstellung von Daten in Dashboards. Es ist der Standard zur Visualisierung von PromQL-Abfragen. Zur vereinfachten Integration der Dienste bietet Grafana einen grafischen Integrationsworkflow für Prometheus [25].

1.4.5 M3DB

M3DB ist eine quelloffene verteilte Timeseriesdatenbank, welche bei dem Onlinevermittlungsunternehmen für Personenbeförderung von Privatpersonen Uber als Alternative zur Datenbank Cassandra entwickelt wurde [6]. Sie ist in der Programmiersprache Golang verfasst und basiert auf einem NOSQL Modell. M3DB ist nach dem Brewers Theorem als eine *CP*-Datenbank [18] einzuordnen, da sie eine Ausfalltoleranz mit dem Replikationfaktor $RF = (N/2) + 1$ (Wobei N die Anzahl der Knoten ist) arbeitet und verschiedene garantierte Konsistenzlevel anbietet. Im Hintergrund verwendet die Datenbank ein externes oder eingebettetes ETCD-Cluster, in welchem die Konfiguration von *Clustermembership*, Namespaces und Platzierung von Shard stattfindet. Die Datenbank implementiert außerdem die Funktion eines *Prometheusremoteprotokollendpunketes* zur Benutzung als Timeseriesdatenbank für Prometheus. Das Management der Datenbank ist über eine REST-API möglich. Die Daten werden von der M3DB in Dateien auf dem Filesystem als sogenannte *Filesets* im eigenen M3TSZ und Protocolbufferformat persistiert. Diese Filesets werden sowohl für die auf Snapshots des `B+-Tree` basierte Storageebene als auch für die Commitlogs verwendet. Nach eigenen Aussagen konnten die gesamten Ausgaben des Monitorings bei Uber durch den Wechsel auf M3DB von zehn auf zwei Prozent reduziert werden [9].

1.4.6 Consul

Consul ist eine von HashiCorp entwickelte verteilte Lösung zur Servicediscovery. Es beinhaltet die Möglichkeit, Dienste über einen internen DNS-Server oder die REST-API abzugreifen. Zur einfachen Visualisierung bietet es eine Weboberfläche.

1.4.7 Ansible

Ansible ist eine von Red Hat entwickelte Lösung zur Automatisierung und einfachen Orchestrierung von IT-Systemen. Es basiert auf einem deklarativen Ansatz, bei welchem der gewünschte Zustand einer Umgebung abstrahiert beschrieben wird. Der Ansibleprozess setzt dann Änderungen implizit auf den Zielsystemen meist über eine einfache Secure Shell (SSH)-Verbindung um, so dass oftmals keine weitere Software als Daemon auf den Zielhosts installiert sein muss. Die Funktionen von Ansible können über Module in Python erweitert werden. Die Informationen über den Zielzustand des Systems werden in einer oder mehreren Dateien im YAML-Format, sogenannten Playbooks, beschrieben (siehe Abbildung 10 auf Seite 20). Um Wiederverwendbarkeit und eine gute Codeübersicht zu garantieren, können Teile eines Playbooks in sogenannte Rollen gebündelt werden. Diese haben das gleiche Format wie Playbooks, sind jedoch, im Gegenteil zu jenen, einer festen Ordnerstruktur unterworfen. Die von Ansible verwalteten Server werden in einem Inventory verwaltet, welches als Liste von Servern im INI- oder YAML-Format vorliegt (siehe Abbildung 7 auf Seite 20). [24]

2 Projektplanung

2.1 Wirtschaftliche Betrachtung

Da es sich bei allen Produkten um freie OpenSource-Software handelt, musste ich keine kostenpflichtigen Lizenzen zum Einkauf übermitteln. Außerdem wurden die benötigten virtuellen Maschinen vom Bund bereitgestellt. Somit werde ich lediglich die beteiligten Mitarbeiter in die wirtschaftliche Betrachtung einfließen lassen *(siehe Tabelle 1 auf Seite 26)*.

2.2 Projektschnittstellen

Die Bereitstellung der virtuellen Maschinen, inklusive des Betriebsystems, wurde von den zuständigen Projektteams des ITZBund durchgeführt. Dies gilt auch für sämtliche Konfigurationen im Bereich der Netzwerke und Firewalls außerhalb der virtuellen Host und ist nicht Teil des Projekts.

2.3 Projektablaufplan

Das Projekt habe ich innerhalb der Regelarbeitszeit zwischen 8:00 Uhr und 18:00 Uhr umgesetzt. Der Projektablaufplan, *siehe Tabelle 2 auf Seite 26*, diente mir hierbei als Grundlage zur Terminierung der einzelnen Aufgaben aller vier Projektphasen. Fremde Projekte, welche in der gleichen Zeitspanne auszuführen waren, habe ich nicht explizit in der Planung visua-

lisiert. Jedoch waren Überschneidungen, zum Beispiel die Bearbeitung von Supportanfragen im Betrieb der SORMAS-Applikation, durchaus im Vorraus miteingeplant, sodass es zu keinen unvorhergesehenen Unterbrechungen des Projektes kam. Pro Arbeitswoche standen mir zwei Arbeitstage für dieses Projekt zur Verfügung, somit erstreckte sich der Bearbeitungszeitraum auf ungefähr zweieinhalb Wochen. Die Aufgaben habe ich den vier Projektphasen *Planung*, *Realisierung*, *Evaluation* und *Dokumentation* zugewiesen.
Bei der *Planung* habe ich mich dazu entschlossen, alle Aufgaben parallel aufzugreifen und zu bearbeiten, um ein möglichst vollständiges Gesamtbild zu erhalten und zielorientiert vorgehen zu können. Im Gegensatz dazu habe ich mich entschieden, die *Realisierung* nacheinander abzuarbeiten, weil die einzelnen Teilschritte eine derartige Abgrenzung erforderten. Mein Ziel war es außerdem, einer Redundanz in der Dokumentation und damit einhergehenden Verwirrungen entgegenzuwirken, weshalb ich die Dokumentation als phasenübergreifendes Element von Anfang bis Projektabschluss einsetzte.

2.4 Durchführung der IST-Analyse

Vor der Durchführung des Projektes bestand das Monitoring der Infrastruktur im ITZBund aus lediglich einem einzelnen virtuellen Server. Die Hardware der virtuellen Maschine war mit 16 GB Hauptspeicher und 4 CPU-Kernen eines *Intel Xenon Gold* der sechsten Generation ausgestattet. Das Speichervolumen der lokalen Festplatte betrug 500 GB auf einem Netzwerkspeicher mit Solid State Drive (SSD)-Basis.
Die minütige Datenmenge *D (MB/min)* des Prometheusservers konnte ich mit der Größe der Metriken D_M *(KB)*, dem Scrapeintervall *t (s)* und der Anzahl der zu überwachenden Server *(n)* kalkulieren.

$$D = 60\frac{n \times D_M}{1000 \times t}$$

Da der bisherige Server mit einem Scrapeintervall von *10* Sekunden arbeitet und die durchschnittlichen Datengrößen des *Nodeexporter 200 KB* der *390 Maschinen* betragen, lässt sich feststellen, dass der Monitoringserver einer Belastung von *468 MB/min* standhalten muss. Aufgrund von fehlender Redundanz ist der Server ein Single Point of Failure im Bereich des Monitorings und der daraus folgenden Alarmierung bei Ausfällen im Service.

2.5 Ermittlung des SOLL-Zustandes

Auf drei virtuellen Maschinen mit derselben Konfiguration (siehe Punkt 2.4) richte ich ein dezentrales Monitoringsystem auf der Basis der Technologien Prometheus und der M3DB ein. Die Implementierung soll einfach *horizontal* skalierbar sein und keinen Single Point of Failure haben. Außerdem ist eine Anforderung, dass alle bisherigen Grafana Dashboards weiter ge-

nutzt werden können. Um nicht auf allen drei Prometheusinstanzen die Konfiguration über alle zu monitorenden SORMAS-Installationen pflegen zu müssen, soll außerdem ein Consulcluster als dynamischer *Service Discovery Provider* eingerichtet werden. Als Basis für die M3DB muss ein ETCD-Cluster eingerichtet werden, welches ich auch auf den Knoten platziere. Außerdem soll auch der Nodeexporter auf dem System installiert sein (siehe Abbildung 1 auf Seite 18). Zur einfachen Wartbarkeit und Installation verwende ich die bereitgestellten Containerimages der Produkte.
Aus Gründen der einfachen Installation, Skalierbarkeit und Wartung verwende ich die in unserem Team oft eingesetzte Automatisierungssoftware Ansible.

2.6 Evaluierung der VMs

Die Mindestgröße eines RAFT-Clusters beträgt drei Mitglieder, um einem sogenannten Split-Brain entgegenzuwirken. Aus diesem Grund werde ich auch dieses Monitoringcluster auf drei Knoten konzipieren (siehe Abbildung 2 auf Seite 18). Bei einem erhöhten Leistungsdruck auf dem System sollen weitere Knoten im Nachhinein hinzugefügt werden können. Als Betriebsysteme können beim ITZBund entweder *RedHat Enterprise Linux* oder *Microsoft Windows Server 2016* verwendet werden. Wegen meiner Expertise und dem Ziel, eine Linuxumgebung zu überwachen, verwende ich ersteres.

3 Projektrealisierung

3.1 Initialisierung des Projektes

Zur Initialisierung des Projektes habe ich auf meinem Arbeitsnotebook zunächst einen Projektordner angelegt, in dem alle dem Projekt zugehörigen Dateien verwaltet werden. Darauffolgend initialisierte ich mit der Versionsverwaltungssoftware *Git* ein *Repository*, um gegebenenfalls auf ältere Versionsstände zugreifen zu können. Für ständige Backups und die Unabhängigkeit von der Zuverlässigkeit meiner Hardware habe ich ein Remoterepository bei der Entwicklerplattform *Github* eingerichtet. Um nicht immer wieder Zeit für multiple Authentifikation zu verlieren, habe ich mich hier für die SSH-Option zur Anmeldung mit meinem *Public-Key* entschieden (siehe Abbildung 11 auf Seite 20). Wie im Punkt 1.4.7 beschrieben benötigt die Automatisierungssoftware *Ansible* eine Inventory-Datei, welche ich in einen Unterordner mit dem Namen *inventory* als gleichnamige INI-Datei platzierte. Da das Projekt aber als Vorlage für mehrere potentielle Installationen dienen soll, habe ich den Pfad der Inventory-Datei in eine versteckte *.gitignore*-Datei im Wurzelverzeichnis des Projekts eingetragen. So wird sie von der Versionsverwaltung nicht beachtet. Als Nächstes habe ich einen Unterordner mit dem Namen *roles* für

die in den nächsten Punkten folgende Ansible Rollen erstellt. Der letzte Teil für die Automatisierung mit *Ansible* war das Erstellen des Playbooks an sich. Dafür habe ich eine Datei mit dem Namen *main.yml* im Projektverzeichnis erstellt. In dieser Datei habe ich das Playbook im YAML Aint Markup Language (YAML)-Format beschrieben (siehe Abbildung 13 auf Seite 21). Das Playbook habe ich mit einer einzigen Task konzipiert, welche die Anforderung stellt, dass die Zielsysteme alle im Array des Unterpunktes *roles* vorhandenen Rollen eingenommen haben. Nun konnte ich das Playbook mit dem *Ansible*-Kommando ausführen (siehe Abbildung 14 auf Seite 21). Da das Kommando mit seinen Optionen kompliziert war und der Anforderung der Einfachheit nicht entsprochen hätte, habe ich mich entschieden, eine Make-Datei im Wurzelverzeichnis des Projekts einzurichten. So konnte ich mit dem simplen Kommando *make ansible.run* das Projekt starten. Außerdem habe ich auf der linken Seite des Befehls in der Makefile die Inventory-Datei als Vorraussetzung eingetragen, damit Make, beim Fehlen dieser, eine Warnung ausgibt und den Vorgang abbricht, ohne den *Ansible*prozess zu starten (siehe Abbildung 15 auf Seite 21).

3.2 Anlegen von Ansible Rollen

3.2.1 Basispakete

Damit *Ansible* alle gewünschten Funktionen ausführen kann, wie zum Beispiel das Management der Firewall der Hosts, müssen vereinzelt Pakete aus den Systemrepositories installiert werden. Um dies auch im gleichen Playbook erledigen zu können, habe ich eine neue Rolle mit dem *ansible-galaxy*-Kommando im *roles*-Ordner initialisiert (siehe Abbildung 12 auf Seite 21). Da diese Rolle gemeinsame Paketinstallationen und Systemänderungen, unabhängig von anderer Software, ausführen lassen sollte, habe ich sie *common* genannt.
Die erste Änderung, die ich dieser Rolle hinzugefügt habe, ist das Setzen des Hostnamens der Maschine auf den Inhalt der Variable `ansible_hostname`, da dieser in der Shell der Systeme vorher nicht angezeigt wurde. Danach habe ich sichergestellt, dass das Paket `epel-release` auf den Systemen installiert ist, welches die Standardpaketquelle für Software außerhalb des Hause Red Hat auf `RHEL`-Derivaten ist. Dieser Schritt war notwendig, um unter anderem Pakete, die bei Netzlink für den Betrieb genutzt werden, herunterladen zu können. Als Nächstes habe ich den Task für die Installation der weiteren Pakete konzipiert (siehe Abbildung 16 auf Seite 21). Der Task benutzt das `dnf`-Modul, Name des Paketmanagers unter `RHEL`-Derivaten, um sicherzustellen, dass die Pakete, welche in der Variable `common_packages` definiert sind, auf den Hosts installiert sind und falls nicht, diese zu installieren. Im Gegensatz zu dem Zustand *(State)* `latest` prüft `present` lediglich die Präsenz der Pakete, ohne sie auf den neusten Stand zu bringen. Da ein unkontrolliertes Update hier fatal wäre, fiel meine Wahl hier auf letztere Option. Es

ist *Common-Practice*, bei Rollen auf die Trennung von Geschäftslogik und Konfigurationsdaten zu achten, um einen möglichst sauberen Code zu erzeugen. Aus diesem Grund habe ich die Variable `common_packages` in der *main.yml* des `defaults`-Ordners deklariert (siehe Abbildung 17 auf Seite 22). Außerdem wird in dieser Rolle im gleichen Stil die Zeitzone auf den Wert *Europe/Berlin* gesetzt und die Firewall über *Systemd* im Startprozess des Betriebsystems aktiviert und auch zur aktuellen Laufzeit gestartet.

3.2.2 Nodeexporter

Um die VMs des Systems selbst überwachen zu können, habe ich den Nodeexporter aus Punkt 1.4.3 auf Seite 3 in das Playbook integriert. Ansible bietet der Interessengemeinschaft *(Community)* mit der *Ansible Galaxy* ein Repository zum Austausch von Rollen an. Dort hat das Projekt *Cloud Alchemy* eine gut gepflegte und weitverbreitete Rolle zur Installation des Nodeexporter zur Verfügung gestellt. Da ich diese Rolle schon in anderen Projekten verwendete, habe ich sie in das Playbook eingefügt.
Dafür habe ich eine Datei mit dem Namen `requirements.yml` im Rollenverzeichnis erstellt und die Rolle als externe Abhängigkeit eingetragen (siehe Abbildung 18 auf Seite 22). Diese Rollen konnte ich dann mit dem `ansible-galaxy`-Kommando herunterladen. Das Herunterladen habe ich als eigene Abhängigkeit in der `Makefile` eingetragen, sodass eine Aktualisierung der Rolle gegebenenfalls mit übernommen werden kann.

3.2.3 ETCD Cluster

Ein ETCD Cluster ist eine Vorraussetzung für die produktive Nutzung der M3DB. Da der ETCD ein persistenter Key-Value-Store ist, habe ich mit dem `file`-Modul einen Ordner zur Speicherung der Daten unter `/var/etcd` vorausgesetzt. Weiter benötigt ein ETCD-Knoten die Öffnung von mindestens drei weiteren Ports, welche ich über das `firewalld`-Modul geöffnet habe. Der nächste Schritt war das Starten des ETCD-Containers mittels der Containerruntime *Podman* und dem gleichnamigen Modul in Ansible. Dafür musste ich in den Modulparametern eine Portweiterleitung der ETCD-Ports auf die Hostmaschine berücksichtigen und als Conatinerimage den String `"quay.io/coreos/etcd:{{ etcd_version }}"` eintragen. Wobei die Variable `etcd_version` aus den *Defaults* der Rolle übernommen wird und das *Tag* mit der Versionsnummer des ETCD darstellt. Von dort wird beim Start des Container das Image heruntergeladen. Als Kommando, welches beim Containerstart ausgeführt wird, habe ich das `etcd`-Programm gesetzt und die nötigen Flags zur Clusterbildung und Portbelegung gesetzt [7] (siehe Abbildung 19 auf Seite 22). Den Namen des Knotens habe ich auf die Variable `ansible_hostname` gesetzt, um im Betrieb nachvollziehen zu können, auf welchem Host sich dieser befindet. Außerdem musste ich die Adresse, auf welcher sich der Knoten bekanntmachen soll, auf die

IP-Adresse des Knotens setzen. Diese konnte ich einfach aus den von Ansible über den Host gesammelten Fakten abgreifen. Als Letztes musste nur das initiale Cluster, bestehend aus allen Knoten, als eine kommaseparierte Liste aus Knotennamen und der zugehörigen URL angeben werden. Dies habe ich mit dem in Ansible inkludierten *Jinja2*-Templating umgesetzt (siehe Abbildung 20 auf Seite 23). Nach der Bildung des Clusters ist noch zu prüfen, ob alle teilnehmenden Knoten im Cluster den Status `healthy` haben. Dazu habe ich Ansible ein Bashskript ausführen lassen, welches mit dem `etcdctl member list`-Kommando (siehe Abbildung 3 auf Seite 18) alle Teilnehmenden samt Status auflistet, mit dem richtigen Status per `grep` filtert, durch `wc` zählt und dann mit der Anzahl der gewünschten Knoten vergleicht. Das Skript wird erst dann beendet, wenn die richtige Anzahl `healthy` ist.

3.2.4 M3DB Cluster

Da alle Voraussetzungen erfüllt sind, kann ich mit der Implementation der M3DB beginnen. Diese ist ebenfalls eine persistente Datenbank und benötigt zwei Verzeichnisse auf der Hostmaschine für die korrekte Funktion. Das eine wird für das Speichern der Snapshots und des Commitlogs, wie in Punkt 1.4.5 auf Seite 3 beschrieben, verwendet. Das andere dient zum *Caching (Zwischenspeicherung)* der *Namespace*-Konfiguration und des *Placing* der Shard im ETCD. Zum Erstellen dieser habe ich das `file`-Modul genutzt. Ein weiteres Verzeichnis enthält eine Konfigurationsdatei im YAML-Format für den Betrieb der M3DB. Diese wird als `config.yml` in den `\etc\m3dbnode`-Ordner eines Knotens aus einer *Jinja2*-Datei des Ansibleprojekts getemplated. Dann werden per `sysctl`-Modul Anpassungen im Kernel zum Dateilimit und Virtualmemory durchgeführt [6]. Als nächstes habe ich wieder über das `firewalld`-Modul die nötigen Ports auf der Hostmaschine geöffnet und über das `podman`-Modul den *m3dbnode*-Container mit Portweiterleitung und Anhang der Verzeichnisse gestartet (siehe Abbildung 21 auf Seite 23). Nachdem das Cluster aufgebaut war, sind zur Nutzung mit Prometheus einige Schritte in der Datenbanklogik zu konfigurieren. M3DB bietet für diese Konfiguration eine REST-Schnittstelle an. Zuerst habe ich das `uri`-Modul verwendet, um einen Namespace mit dem Namen *default* im M3DB-Cluster, per JSON-Payload zu initialisieren. Die nächste Konfiguration, die ich getätigt habe, ist die der Platzierung der Shards auf die Knoten des Clusters. Dazu habe ich die Anzahl der Shards auf 32 und den Replikationfaktor, aufgrund der Konfigurationsempfehlung der M3DB-Entwickler*innen zur Produktion, auf drei gesetzt (siehe Abbildung 22 auf Seite 24). Als letzten Schritt der Rolle muss ich Ansible angeben, auf die Fertigstellung des *Bootstrap*-Prozesses zu warten, bis dem `uri`-Modul am REST-Endpoint `/api/v1/services/m3db/placement` ein JSON-Body übermittelt wird, in dem alle Shards den Status `AVAILABLE` haben. So wird sichergestellt, dass vorher keine anderen Systeme auf eine unvorbereitete Datenbank zugreifen.

3.2.5 Consul Cluster

Um neue Services über eine API hinzufügen zu können, habe ich ein Consulcluster zur dynamischen Servicediscovery errichtet. Mein erster Schritt war wieder die Öffnung der von Consul genutzten Ports in der Systemfirewall per `firewalld`-Modul. Auch Consul verwendet zur Persistierung Verzeichnisse auf dem Dateisystem, welche ich mit dem `file`-Modul per Ansible voraussetzte.
Zum Start des Clusters muss zuerst eine einzige initiale Consulinstanz als Podmancontainer gestartet habe. Zwei weitere Knoten wurden als Voraussetzung für den Betrieb konfiguriert. Danach habe ich einen Task zum Start aller anderen Clustermitglieder per Podman konzipiert und warte als nächsten Schritt in der Rolle auf die Vollständigkeit des Clusters. Dies geschieht ähnlich wie im Kapitel 3.2.3 auf Seite 8 zum ETCD beschrieben.

3.2.6 Prometheus

Auf jedem Host wird ein eigenständiger Prometheusserver installiert, welcher die Metriken der Metricsexporter einsammelt und in das gemeinsame M3DB-Cluster über die Remote-Write-API einpflegt. Dazu habe ich zuerst ein Jinja2-Template der `prometheus.yml` als Konfigurationsdatei per `template`-Modul auf alle Hosts persistiert (siehe Abbildung 23 auf Seite 24). Als Nächstes habe ich den Port `9090` für Prometheus in der Systemfirewall per `firewalld`-Modul geöffnet und den Prometheuscontainer über das `podman`-Modul mit der Konfigurationsdatei gestartet. [19]

3.2.7 Grafana

Da Grafana als Visualisierungswerkzeug keine betriebskritische Anwendung ist, habe ich die Applikation nur auf dem ersten Host des Clusters platziert. Dafür habe ich den Port 3000 in der Firewall per `firewalld`-Modul geöffnet und den Grafanacontainer per `podman`-Modul gestartet.

3.3 Installation der Software

3.3.1 Vorbereitung der Installation

Zuerst habe ich sichergestellt, dass alle Rollen in das Playbook übernommen sind und daraufhin das Inventory mit den IP-Adressen der Hosts unter der Gruppe `[nodes]` befüllt.

3.3.2 Durchführen der Installation

Zur Durchführung der Installation auf den Hosts des Inventory habe ich den Ansibleprozess über `make` gestartet, was zur Ausgabe in der Abbildung 5 auf Seite 19 führte. Der Prozess

dauerte ungefähr eine halbe Stunde, wobei der `Bootstrap` der M3DB die meiste Zeit beanspruchte.
Nachdem der Prozess beendet war, wurde von Ansible noch eine kurze Zusammenfassung der Schritte aufgeführt (siehe Abbildung 4 auf Seite 18).

3.4 Eintragung von Betriebsdaten und Dashboards

In der Nachkonfiguration habe ich folgende Schritte durchgeführt, welche sich per Ansible als schwierig dargestellt hätten, da es sich um umgebungsspezifische Anpassungen handelt. Um einem Bruch mit der Anforderung der einfachen Replikation auf andere Umgebungen zu verhindern, habe ich deshalb das Hinzufügen der von Netzlink genutzten Dashboards manuel durchgeführt und einige von einem Onlineservice, gestellt durch *Grafana Labs*, importiert. Daraufhin habe ich einige Services in das Consulcluster hinzugefügt, welche dann auch im Prometheus als *Target* auftauchten und deren Betriebsdaten sich über PromQL-Abfragen abrufen ließen.

3.5 Test der Monitoringlösung

Die ersten Maßnahmen, die ich durchgeführt habe, waren das Überprüfen der einzelnen Services über die Weboberflächen beziehungsweise der Kommandozeile. Alle waren insgesamt in einem funktionstüchtigen Zustand. Nachdem dies sichergestellt war, habe ich die Daten der bisher eingefügten Services über die Prometheusoberfläche und die eingespielten Grafanadashboards darstellen lassen (siehe Abbildung 6 auf Seite 19). Danach war nur noch die Partitionstoleranz gegenüber Ausfällen von Knoten überprüfen.
Dies habe ich durch das Abschalten der Knoten simuliert. Bei einem Ausfall von einem von drei Knoten konnten alle Services weiter benutzt werden, wobei im Grafana-Dashboard der M3DB eine Unterreplizierung der *Datenbank* zu erkennen war. Beim Abschalten des zweiten Knotens versagte der Consul aufgrund des fehlenden Clusters. Das gesamte System war beim Anschalten der Maschinen nach einer kurzen Replikationsphase der Datenbank wieder voll funktionsfähig. Die Testphase konnte somit erfolgreich abgeschlossen werden.

4 Fazit

Als Ergebnis kann ich feststellen, dass durch die Verteilung von Betriebsdaten auf mehrere Host ein lückenloser Betrieb der Plattform möglich ist. Zur Nutzung der M3DB in kleineren Umgebungen bei anderen Projekten ist noch anzumerken, dass für den Vorteil der Partitionstoleranz wegen des Overheads von mindestens zwei weiteren VMs im Vergleich zu einem einzelnen Prometheus ein höherer wirtschaftlicher Aufwand zu tätigen ist [10]. Entsprechend

sollte eine Kalkulation gemacht werden, ob der Ausfall des Monitorings bei nur einer Instanz die erhöhten Kosten eines Clusters rechtfertigt.
Desweiteren bin ich von der ursprüngliche Planung im GANTT-Diagram auf Seite 26 am zweiten Tag ein wenig abgewichen, da ich für die Umsetzung der ETCD-Rolle lediglich eine Stunde brauchte. Diese Zeit kam mir bei der Konstruktion der M3DB-Rolle zugute, da die Dokumentation der Datenbank einige Lücken aufwies. Durch die Mithilfe meines Teamleiters und der M3DB-Interessengemeinschaft *(Community)* konnte ich diese gut überbrücken. Ich hatte während meiner Ausbildung bei Netzlink noch keine Infrastrukturautomatisierung in dieser Dimension umgesetzt. Die Nutzung der Containerimages und die Einteilung in Rollen als Abstraktionsebene hat einen großen Teil der Komplexität ausgeblendet. So konnte ich mich immer auf die Bearbeitung einer Teilaufgabe konzentrieren. Ich innerhalb des Projekts gelernt, wie die Konfiguration einer produktiven Infrastruktur automatisiert wird und einen Einblick in die behördlichen Arbeitsweisen unserer Kunden gewonnen.

Glossar

Availability Management Ein Unterpunkt von ITIL bei dem die Verfügbarkeit eines Dienstes anhand von Qualitätsparametern definiert wird. 1

Brewers Theorem Sagt aus, dass die Garantie, in einer verteilten Software Ausfalltoleranz, Verfügbarkeit und Konsistenz gleichermaßen zu berücksichtigen, nicht möglich ist [18]. 3, 15

Bundesministerium für Gesundheit Eine Behörde auf Bundesebene mit dem Zuständigkeit für das deutsche Gesundheitswesen [1]. 1, 15

Cassandra Eine bei Facebook entwickelte, freie und OpenSource NOSQL-Datenbank. Cassandra wird meistens verteilt betrieben und macht dies durch eine *Distributed Hashtable (DHT)* möglich. Die Daten werden als *Spalten* in Tabellen organisiert dargestellt und können mittels der *Cassandra Query Language (CQL)* abgefragt werden [11]. 3

Christlich Demokratische Union Eine christdemokratische Partei aus Deutschland, im politischen Spektrum mitte-rechts angeordnet. 1, 16

Containerimages Ein für Softwarecontainer verwendetes unveränderliches Paketformat mit den Dateien der Anwendungen und der Laufzeitumgebung. 5, 8, 11

Coronakrise Begriff der Medien für die soziale und ökonomische Krisensituation in Folge der pandemische Ausbreitung des 2019 entdeckten Coronavirus SARS-CoV-2. 1, 2

Daemon Eine etablierte Bezeichnung für Server- oder Hindergrundprozesse auf UNIX und seinen Nachfolgern. 3

Day2Ops Ein zusammenfassender Begriff für alle Operationen, welche nach der Inbetriebnahme von Softwaresystemen ausgeführt werden. Eingeführt wurde der Begriff von der Firma D2IQ (ehemals Mesosphere) [17]. 2

Docker Ein Unternehmen, welches die gleichnamige OpenSource Containerruntime entwickelt und Softwarecontainerisierung in das betriebliche Standardportfolio einführte [16]. 2

Domain Specific Language Eine nach Zeitpunkten indizierte Serie von Datenpunkten. 2, 16

ETCD Ein weitverbreiteter verteilter Key-Value-Store basierend auf dem Consensusprotocol *RAFT* [4]. 3

Golang Eine bei Google unter anderem von Ken Thompson entwickelte statischtypisierte Systemprogrammiersprache mit einer automatischen Speicherbereinigung (Garbage Collection) und Schwerpunkt auf einen minimalen Syntax basierend auf der Sprache C [15]. 2, 3

INI Ein menschenlesbares, einfaches Textformat, bei dem Daten als Schlüssel-Werte-Paare formatiert sind. 3, 6, 14, 17, 20

Inventory Eine Liste von Severn im YAML- oder INI-Format für die Verwaltung mit Ansible. Server können auch gruppiert werden. 3, 6, 10, 17, 20

ITIL Leitfaden und Standard im IT-Service-Management. 1, 13, 16

ITZBund Eine Behörde, welche IT-Dienstleistungen für staatliche Behörden der Bundesregierung erbringt [13]. 1, 2, 4, 5, 6, 16, 26

JSON Eine einfache Datenbeschreibungssprache mit dem Ursprung aus der Notation von JavaScript Objekten. 9, 16

Make Ein im Unixumfeld weitverbreitetes Werzeug zum Management und Vereinfachen von Softwarebuilds (wie Kompilierung und Linken von Programmen in C). 6, 17, 20

Metricsexporter Ein Programm zum Sammeln von Betriebsdaten und Übertragung in Prometheusmetriken, welche von einem HTTP-Endpunkt abgerufen werden können [20]. 2, 10

Microservice Eine Softwarearchitektur, in welcher ein Dienst aus mehreren kleineren Diensten gebildet wird, welche jedoch unabhängig voneinander betrieben werden können. Es wird so erhofft, weniger Entwicklungsbedarf und bessere Lesbarkeit des Quellcodes zu erreichen [21]. 2

Namespaces Höchste Abstraktionseinheit der M3DB. Vergleichbar mit den Datenbanken der relationalen Datenbankmanagementsoftware. 3, 9

NOSQL Ein Überbegriff für alle Datenbankkonzepte, die nicht dem relationalem SQL-Ansatz folgen [22]. 3

Pandemiemanagementsoftware Softwaresystem zur Unterstützung des Gesundheitssystems im Zuge von Kontaktverfolgung, Quarantänemanagement etc. 1, 2, 16

Playbooks Eine oder mehrere Dateien im YAML-Format, welche den Zielzustand eines von Ansible verwalteten Systems beschreiben. 3, 6, 7, 8, 15, 17, 20, 26

Protocolbuffer Ein von Google entwickelter Standard zur nachhaltigen Darstellung von Daten im Binärformat. 3

Python Eine beliebte objektorientierte, dynamischtypisierte Skriptsprache mit einfachem Syntax. 3

RAFT Ein Konsensalgorithmus mit dem Ziel der einfachen Verständlichkeit [5]. 6

Remoterepository Im Netzwerk oder Online vorbehaltene Kopien eines git-Repository. 6, 17, 20

Rollen Ein hierarchisch untergeordneter Teil eines Playbooks, welcher zur Übersichtlichkeit und Wiederverwendbarkeit von Code eines Ansibleprojektes beitragen sollen. 3, 6, 7, 8, 9, 10, 11, 17, 20, 26

Shard Eine virtuelle Einheit der M3DB, in welcher Timeseriesschlüssel auf einen Knoten zugewiesen werden. 3, 9

Single Point of Failure Ein Teil eines Systems, welches keine Ausfalltoleranz aufweist und das gesamte System gefährden kann. 5

Split-Brain Ein Problem in Computerclustern mit gerader Knotenanzahl, bei dem die Knoten in gleichgroße Gruppen aufgetrennt werden und keine Mehrheit für die Aufrechterhaltung der Konsistenz gebildet werden kann (siehe Brewers Theorem) [12]. 6

SSH Ein Netzwerkprotokoll zur verschlüsselten Kommunikation über Netzwerke. Meist wird das Protokoll für die Verbindung auf das Terminal eines Servers verwendet. Die bekannteste Implementation ist die freie Software *OpenSSH*. Die Portnummer von SSH ist die 22. 3, 16

Task Eine Anforderung an ein System, die Ansible ausführen soll und kleinste Einheit eines Playbooks. 6, 7, 17, 20

Timeseries Eine nach Zeitpunkten indizierte Serie von Datenpunkten. 2, 3, 15

Whiteboxmonitoring Eine Monitoringpraxis, wobei die zu monitorende Applikation selbst die eigenen Betriebsdaten für das Monitoringsystem bereitstellt [14]. 2

YAML Eine einfache Datenbeschreibungssprache, bei der die Einrückung die Datenstruktur wiedergibt. 3, 6, 14, 16

Akronyme

BMG Bundesministerium für Gesundheit. *Glossar:* Bundesministerium für Gesundheit, 1, 2

CDU Christlich Demokratische Union . *Glossar:* Christlich Demokratische Union, 1

DSL Domain Specific Language . *Glossar:* Domain Specific Language, 2

EBNF extended Backus-Naur Form. 2

HTTP Hypertext Transfer Protocol. 2, 14, 17, 20

HZI Helmholtz Zentrum für Infektionsforschung. 2

ITIL Information Technology Infrastructure Library . *Glossar:* ITIL

ITZBund Informationstechnikzentrum Bund . *Glossar:* ITZBund

JSON JavaScript Object Notation . *Glossar:* JSON

Netzlink Netzlink Informationstechnik GmbH. 1, 2, 7, 11

PromQL Prometheus Query Language. 2, 3, 11, 17, 20

REST Representational State Transfer. 2, 3, 9

SaaS Software as a Service. 1

SORMAS ”Surveillance Outbreak Response Management and Analysis System” . *Glossar:* Pandemiemanagementsoftware, 1, 2, 4, 5

SSD Solid State Drive. 5

SSH Secure Shell . *Glossar:* SSH, 3, 6

VM virtuelle Maschine. , 5, 6, 11

YAML YAML Aint Markup Language . *Glossar:* YAML, 6, 9, 16

Abbildungsverzeichnis

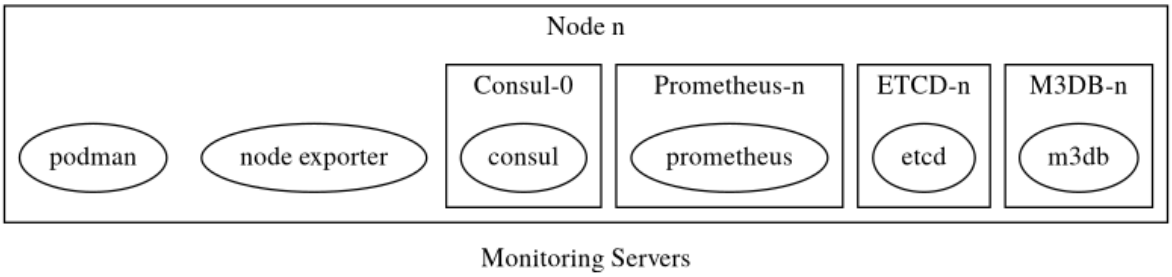

Abbildung 1: Diagramm zur Übersicht eines Monitoringknotens

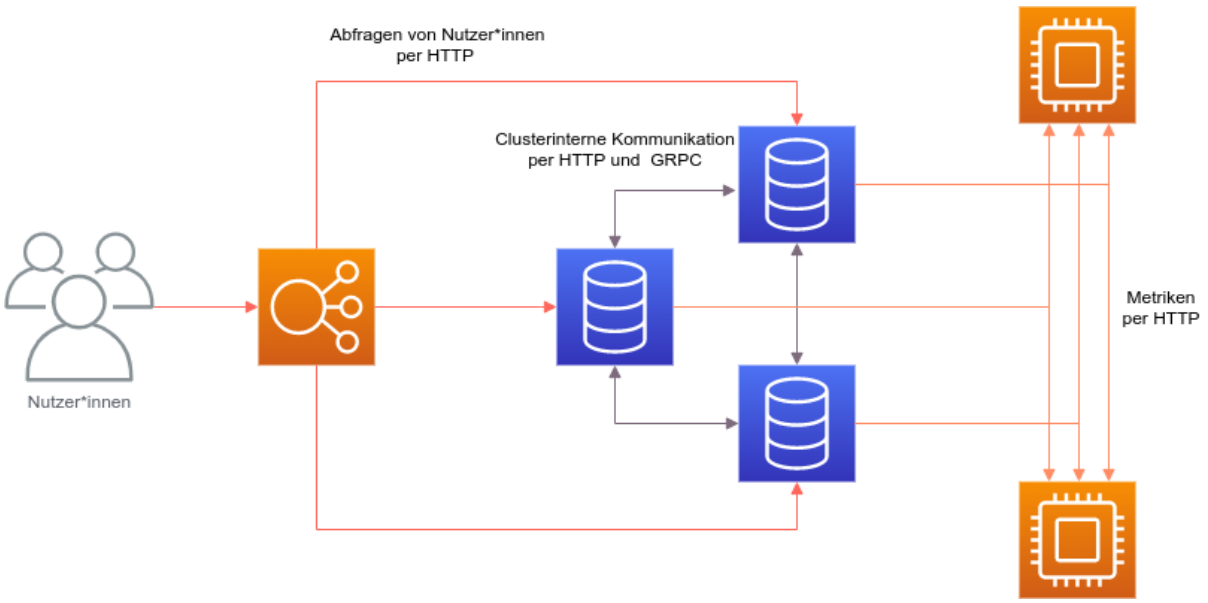

Abbildung 2: Diagramm zur Übersicht des Netzwerkes

```
[root@m3db-node-0 ~]# etcdctl member list
3ca112245806383d, started, m3db-node-0, http://206.189.58.96:2380, http://206.189.58.96:2379,http://206.189.58.96:4001
6cd6602b870c398e, started, m3db-node-2, http://206.81.27.220:2380, http://206.81.27.220:2379,http://206.81.27.220:4001
7671f0b96c2b7f83, started, m3db-node-1, http://178.128.199.150:2380, http://178.128.199.150:2379,http://178.128.199.150:4001
```

Abbildung 3: Ausgabe des ETCD zum Zustand des Cluster

```
PLAY RECAP *********************************************************************
178.128.199.150            : ok=48   changed=13   unreachable=0    failed=0    skipped=19   rescued=0    ignored=0
206.189.58.96              : ok=58   changed=18   unreachable=0    failed=0    skipped=19   rescued=0    ignored=0
206.81.27.220              : ok=48   changed=13   unreachable=0    failed=0    skipped=19   rescued=0    ignored=0
```

Abbildung 4: Ausgabe der Zusammenfassung des Ansibleprozess

```
TASK [m3db : Create namespace with config] ***************************
ok: [167.99.247.8]

TASK [m3db : This step may take an incredible amount of time.] *******
changed: [167.99.247.8]

TASK [m3db : Init placement with config] *****************************
ok: [167.99.247.8]

TASK [m3db : Create database in M3DB] ********************************
ok: [167.99.247.8]

TASK [m3db : Fail if error] ******************************************
skipping: [167.99.247.8]
skipping: [167.99.251.239]
skipping: [167.99.247.55]

TASK [m3db : Wait for readiness in namespace] ************************
skipping: [167.99.251.239]
skipping: [167.99.247.55]
FAILED - RETRYING: Wait for readiness in namespace (120 retries left).
ok: [167.99.247.8]

TASK [prometheus : Create the config] ********************************
changed: [167.99.247.8]
changed: [167.99.251.239]
changed: [167.99.247.55]

TASK [prometheus : Open port] ****************************************
changed: [167.99.247.8]
changed: [167.99.251.239]
changed: [167.99.247.55]

TASK [prometheus : Start prometheus container] ***********************
```

Abbildung 5: Ausgabe des Ansibleprozess

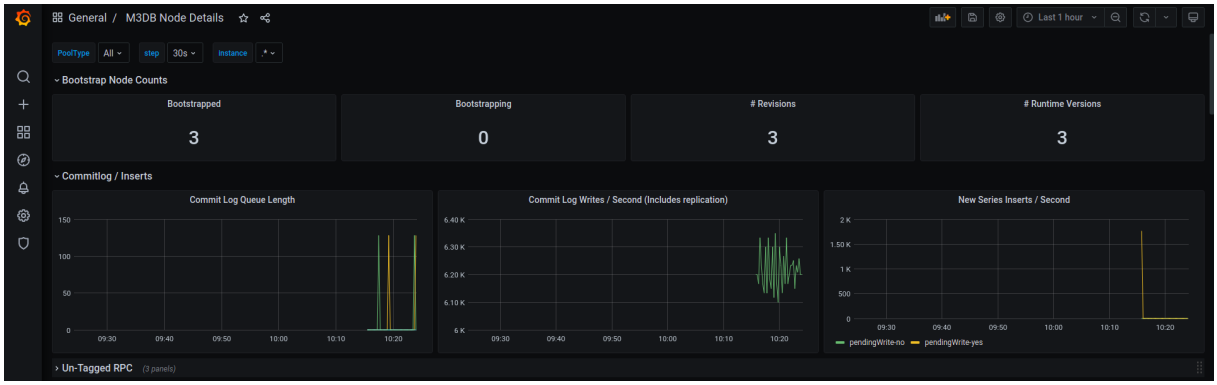

Abbildung 6: Grafana M3 Dashboard

```
[databse_servers]
165.xxx.87.xxx
161.xxx.213.xxx
165.xxx.172.xxx
```

Abbildung 7: Beispiel für ein Inventory im INI-Format

```
# HELP http_requests_total The total number of HTTP requests.
# TYPE http_requests_total counter
http_requests_total{method="post",code="200"} 678 1925066363000
http_requests_total{method="post",code="400"} 1 1647066363000
```

Abbildung 8: Metricsformat Beispiel eines Webservers

```
sum(rate(http_requests_total{job="prometheus"}[5d]))
```

Abbildung 9: PromQL für die Summe aller HTTP-Request mit dem Label *Prometheus* der letzten fünf Tage

```
---
- name: Postgres Playbook
  hosts: database_servers
  become: yes
  tasks:
  - name: ensure postgres is installed
    dnf:
      name: postgresql-server
      state: present
```

Abbildung 10: Playbook zur Installation einer *Postgres*-Datenbankmanagementsoftware

```
git init
git remote add origin git@github.com:nk-designz/distributed_prometheus.git
```

Abbildung 11: Initialisierung des git-Repository mit Remoterepository bei *Github*

```
ansible-galaxy init common
```

Abbildung 12: Initialisierung der *common*-Rolle

```
---
- name: Setup Distributed Prometheus
  hosts: all
  become: yes
  remote_user: root
  roles:
  - cloudalchemy.node_exporter
```

Abbildung 13: *main.yml* mit der *NodeExporter*-Rolle

```
ANSIBLE_HOST_KEY_CHECKING=false \
ansible-playbook \
-e 'ansible_python_interpreter=/usr/bin/python3' \
-i inventory/hosts.ini \
main.yml
```

Abbildung 14: Kommando zum Ausführen des Playbooks

```
ansible.run: inventory/hosts.ini
      ANSIBLE_HOST_KEY_CHECKING=false \
      ansible-playbook \
      -e 'ansible_python_interpreter=/usr/bin/python3' \
      -i inventory/hosts.ini \
      main.yml
```

Abbildung 15: Eintrag in der Make-Datei zum Ausführen des Projekts

```
- name: Install common packages
  dnf:
    name: "{{ common_packages }}"
    state: present
```

Abbildung 16: Installation von allgemeinen Paketen

```
# common packages for the nodes
common_packages:
- vim
- htop
- podman
- firewalld
- python3-firewall
- jq
```

Abbildung 17: Allgemeine Pakete im *defaults*-Verzeichnis

```
---
- src: https://github.com/cloudalchemy/ansible-node-exporter
  name: cloudalchemy.node_exporter
  scm: git
  version: master
```

Abbildung 18: Externe Rollen in der *requirements.yml*

```
command: |
  /usr/local/bin/etcd
  --name {{ ansible_hostname }}
  --data-dir /var/run/etcd
  --advertise-client-urls http://{{ etcd_ip }}:2379,http://{{ etcd_ip }}:4001
  --listen-client-urls http://0.0.0.0:2379,http://0.0.0.0:4001
  --initial-advertise-peer-urls http://{{ etcd_ip }}:2380
  --listen-peer-urls http://0.0.0.0:2380
  --initial-cluster-token dont-use-this-token
  --initial-cluster {{ etcd_initial_cluster }}
  --initial-cluster-state new
  --data-dir /var/run/etcd
```

Abbildung 19: Ausschnitt aus dem Task zum Starten des ETCD-Container

```
etcd_ip: "{{ hostvars[inventory_hostname]['ansible_default_ipv4']['address'] }}"
etcd_initial_cluster: "{% for host in groups['nodes'] %}{{
    hostvars[host]['ansible_facts']['hostname'] }}=http://{{
    hostvars[host]['ansible_facts']['eth0']['ipv4']['address'] }}:2380{% if not
    loop.last %},{% endif %}{% endfor %}"
# etcd0=http://X.X.X.X:2380,etcd1=http://X.X.X.Y:2380,etcd2=http://X.X.X.Z:2380
```

Abbildung 20: Setzen der IP- und Clustervariablen aus den *Absible Facts*

```
volume:
- /etc/m3dbnode:/etc/m3dbnode
- /var/m3db:/var/lib/m3db
- /var/m3kv:/var/lib/m3kv
publish:
- "7201:7201"
- "7203:7203"
- "9000:9000"
- "9001:9001"
- "9002:9002"
- "9003:9003"
- "9004:9004"
```

Abbildung 21: Ausschnitt des Task zum Start des *m3db*-Containers

```
{
  "num_shards": 32,
  "replication_factor": 3,
  "instances": [
{% for host in groups['all'] %}
   {
     "id": "{{ hostvars[host].ansible_facts.hostname }}",
     "isolation_group": "{{ hostvars[host].ansible_facts.hostname }}_isolation_group",
     "zone": "embedded",
     "weight": 100,
     "endpoint": "{{ hostvars[host].ansible_facts.default_ipv4.address }}:9000",
     "hostname": "{{ hostvars[host].ansible_facts.hostname }}",
     "port": 9000
   },
{% endfor %}
  ]
}
```

Abbildung 22: Template für die Payload zur *Placement*-Konfiguration

```
    global:
      scrape_interval: 15s
      evaluation_interval: 15s
    remote_write:
    - url: "http://localhost:7201/api/v1/prom/remote/write"
    scrape_configs:
    - job_name: 'consul_services'
    consul_sd_configs:
    - server: "127.0.0.1:8500"
      datacenter: "{{ consul_datacenter }}"
      scheme: http
      refresh_interval: "60s"
```

Abbildung 23: Ausschnitt aus der Prometheuskonfiguration

Tabellenverzeichnis

Phasen	Aufwand *(Stunden)*	Kosten *(EUR)*
Planung	5	300
Realisierung	19	1140
Evaluation	3	180
Dokumentation	8	480
Gesamt	35	2.100

Ressource	Anzahl	Kosten *(EUR)*
VServer	3	0
Reverseproxy	2	0
Gesamt		0

Ressourcen beim ITZBund

Tabelle 1: Aufwandsschätzung aller Kosten bezogen auf die Projektphasen

Abschnitte	Stunden	1	2	3	4	5
Durchführung der IST-Analyse	*1h*					
Ermittlung des Soll Zustands	*2h*					
Evaluierung der Spezifikation der VMs	*2h*					
Aufsetzen der Projektstruktur des Ansible Playbooks	*1h*					
Konstruktion eines Playbooks zum Einrichtung des Monitorings	*2h*					
Konstruktion einer Rolle für einen ETCD-Knoten	*3h*					
Konstruktion einer Rolle für einen M3DB-Knoten	*4h*					
Konstruktion einer Rolle für einen Consul-Knoten	*3h*					
Konstruktion einer Rolle für einen Prometheus-Host	*3h*					
Konstruktion einer Rolle für einen Grafana-Host	*2h*					
Durchführung der Installation	*1h*					
Eintragen von Betriebsdaten und Dashboards	*1h*					
Test der Monitoringlösung	*2h*					
Dokumentation	*8h*					
Tagesstunden		8h	8h	8h	8h	3h
Gesamtstunden				35h		

Tabelle 2: GANTT-Diagramm der Projektplanung

Projektplanung
Realisierung
Evaluation
Dokumentation

Legende des GANTT-Diagramm

Literatur

[1] https://www.bundesgesundheitsministerium.de/

05.03.2021 8:43 Uhr

[2] https://www.sormas-oegd.de/

05.03.2021 8:45 Uhr

[3] https://www.itzbund.de/

05.03.2021 9:21 Uhr

[4] https://etcd.io/

09.03.2021 8:20 Uhr

[5] https://raft.github.io/raft.pdf

09.03.2021 8:33 Uhr

[6] https://m3db.io

09.03.2021 15:02 Uhr

[7] https://etcd.io/docs/v3.4/op-guide/configuration/

09.03.2021 9:40 Uhr

[8] https://m3db.io/docs/operational_guide/placement_configuration/

09.03.2021 16:01 Uhr

[9] https://eng.uber.com/m3/

09.04.2021 16:20 Uhr

[10] https://www.youtube.com/watch?v=CcH13GyszHI

09.04.2021 16:15 Uhr

[11] https://cassandra.apache.org/

05.03.2021 13:23 Uhr

[12] https://de.wikipedia.org/wiki/Split_Brain_(Informatik)

09.04.2021 15:32 Uhr

[13] https://www.itzbund.de/DE/home/home_node.html

09.04.2021 12:08 Uhr

[14] https://sre.google/sre-book/monitoring-distributed-systems/

09.04.2021 7:24 Uhr

[15] Introducing Go - Caleb Doxsey ISBN: 9781491941959
2016

[16] Docker - Sean P. Kane, Karl Matthias ISBN: 9781492036739
2018

[17] https://www.youtube.com/watch?v=gqwcUgZOoyI
09.04.2021 14:08 Uhr

[18] https://www.ibm.com/cloud/learn/cap-theorem
09.03.2021 11:31 Uhr

[19] https://prometheus.io/docs/
14.03.2021 15:08 Uhr

[20] https://prometheus.io/docs/instrumenting/exporters/
14.03.2021 16:22 Uhr

[21] https://microservices.io/
09.04.2021 14:17 Uhr

[22] https://aws.amazon.com/de/nosql/
09.04.2021 11:00 Uhr

[23] https://docs.ansible.com/ansible/latest/user_guide/playbooks.html
09.04.2021 10:01 Uhr

[24] https://www.ansible.com/resources/whitepapers/ansible-in-depth
09.04.2021 11:14 Uhr

[25] https://grafana.com/docs/
09.04.2021 12:01 Uhr

[26] https://podman.io/
09.04.2021 10:01 Uhr

[27] ITIL Foundation: ITIL 4 Edition - AXELOS ISBN: 9780113316144
2019

Anlagen

Genehmigter Projektantrag

Bestätigung über die durchgeführte Projektarbeit

IHK Braunschweig

Sommerprüfung 2021

Ausbildungsberuf
Fachinformatiker/-in Systemintegration

Prüfungsbezirk
Braunschweig FISYS 11975 (AP T2V1)

Herr Nico Kahlert
Identnummer: 594872
E-Mail: nka@netzlink.com, Telefon: +49 175 2818906

Ausbildungsbetrieb: Netzlink Informationstechnik GmbH
Projektbetreuer: Herr Tom Hutter
E-Mail: toh@netzlink.com, Telefon: +49 531 7073430

Thema der Projektarbeit
Einrichtung einer verteilten Monitoringlösung auf Basis von Prometheus und M3 im Bereich des behördlichen Gesundheitswesen.

1 Thema der Projektarbeit

Einrichtung einer verteilten Monitoringlösung auf Basis von Prometheus und M3 im Bereich des behördlichen Gesundheitswesen.

2 Geplanter Bearbeitungszeitraum

Beginn: 16.02.2021
Ende: 28.04.2021

3 Projektbeschreibung

Das Projekt ”Einrichtung einer verteilten Monitoringlösung auf Basis von Prometheus und M3 im Bereich des behördlichen Gesundheitswesen” führe ich im Auftrag meines Ausbildungsbetriebs, der Netzlink Informationstechnik GmbH, durch.

Ziel des Projektes ist es, eine verteilte Monitoringlösung aufzusetzen, die Metriken mehrerer tausend Endpunkte sammeln, speichern und abfragen kann.
Da das System deklarativ horizontal skalierbar sein soll und gegebenenfalls in anderen Umgebungen repliziert werden soll, werde ich den Aufbau mittels des Konfigurationsmanagementsystems ”Ansible” von RedHat beschreiben und später auch darüber ausrollen.
Als Basis für das System werden zunächst drei virtuelle Maschinen mit dem Betriebssystem Redhat Enterprise Linux dienen, welche vom ITZBund vorbereitet werden.
Innerhalb der Realisierung werde ich die Containervirtualisierungssoftware Docker installieren und darauf die Komponenten der M3 Metrikplattform als Container starten, welche vorher als Cluster zu konfigurieren sind. Folgend werde ich den Prometheus starten und für die Auslagerung der Schreib- und Lesevorgänge auf die M3 konfigurieren.
Zum Abschluss werde ich mehrere Metrikendpunkte in das System eintragen und es als Datenquelle in ein Grafana eintragen.

4 Projektumfeld

Das Projekt führe ich für meinen Ausbildungsbetrieb, die Firma Netzlink Informationstechnik GmbH (nachfolgend Netzlink genannt), im Bereich der Abteilung ”OpenSolutions” durch. Die Abteilung ”OpenSolutions” ist verantwortlich für die Planung und den Betrieb von OpenSource-Technologien, sowohl im eigenen Rechenzentrum, als auch in Kundenumgebungen. Netzlink hat 90 Mitarbeiter an drei Standorten verteilt und betreibt drei georedundante Rechenzentren im Großraum Braunschweig, Hannover und Salzgitter. Für das Helmholtzzentrum für Infektionsforschung in Braunschweig betreibt Netzlink eine Applikation zur epidemiologischen Kontaktverfolgung für die deutschen Gesundheitsämter in den georedundanten Rechenzentren des ITZBund. In jenen Rechenzentren wird auch das in diesem Projekt beschriebene Monitoringsystem betrieben.

5 Projektphasen mit Zeitplanung

1 Planung (5 Stunden)
1.1 Durchführung der IST-Analyse (1 Stunde)
1.2 Ermittlung des SOLL-Zustands (2 Stunden)
1.3 Evaluierung der Spezifikationen der VMs (2 Stunden)
2 Realisierung (19 Stunden)
2.1 Aufsetzen der Projektstruktur des Ansible Playbooks (1 Stunde)
2.2 Konstruktion einer Ansible Rolle für einen ETCD-Knoten (3 Stunden)
2.3 Konstruktion einer Ansible Rolle für einen M3-Knoten (4 Stunden)
2.4 Konstruktion einer Ansible Rolle für einen Consul-Knoten (3 Stunden)
2.5 Konstruktion einer Ansible Rolle für einen Prometheus-Host (3 Stunden)
2.6 Konstruktion einer Ansible Rolle für einen Grafana-Host (2 Stunde)
2.7 Konstruktion eines Ansible Playbooks zur Einrichtung des Monitorings (2 Stunden)
2.8 Ausführung der Installation (1 Stunde)
3 Evaluation (3 Stunden)
3.1 Eintragen von Metrikendpunkten und Dashboards (1 Stunde)
3.2 Test des Monitorings (2 Stunden)
4 Dokumentation (8 Stunden)
4.1 Projektdokumentation (8 Stunden)

6 Dokumentation zur Projektarbeit

Die Dokumentation werde ich als prozessorientierten Bericht verfassen. Dabei werde ich mich an den in Punkt fünf beschriebenen Projektphasen orientieren. Teil der Dokumentation wird ein Projektablaufplan sein.

7 Anlagen

keine

8 Präsentationsmittel

- Notebook
- Beamer
- Presenter

9 Hinweis!

Ich bestätige, dass der Projektantrag dem Ausbildungsbetrieb vorgelegt und vom Ausbildenden genehmigt wurde. Der Projektantrag enthält keine Betriebsgeheimnisse. Soweit diese für die Antragstellung notwendig sind, wurden nach Rücksprache mit dem Ausbildenden die

entsprechenden Stellen unkenntlich gemacht.

Mit dem Absenden des Projektantrages bestätige ich weiterhin, dass der Antrag eigenständig von mir angefertigt wurde. Ferner sichere ich zu, dass im Projektantrag personenbezogene Daten (d. h. Daten über die eine Person identifizierbar oder bestimmbar ist) nur verwendet werden, wenn die betroffene Person hierin eingewilligt hat.

Bei meiner ersten Anmeldung im Online-Portal wurde ich darauf hingewiesen, dass meine Arbeit bei Täuschungshandlungen bzw. Ordnungsverstößen mit „null“ Punkten bewertet werden kann. Ich bin weiter darüber aufgeklärt worden, dass dies auch dann gilt, wenn festgestellt wird, dass meine Arbeit im Ganzen oder zu Teilen mit der eines anderen Prüfungsteilnehmers übereinstimmt. Es ist mir bewusst, dass Kontrollen durchgeführt werden.

IHK Braunschweig

Braunschweig | Salzgitter | Wolfenbüttel | Goslar | Peine | Helmstedt

Bestätigung über die durchgeführte Projektarbeit

(Diese Bestätigung ist mit der Projektdokumentation einzureichen!)

Berufsbezeichnung/Fachrichtung/Einsatzgebiet/Fachbereich:
Fachinformatiker für Systemintegration

Prüfungsteilnehmer/-in: Nico Kahlert

Ausbildungsbetrieb/Umschulungsträger: Netzlink Informationstechnik GmbH

Azubi-Identnummer:

Abschlussprüfung ☒ **Sommer** ○ **Winter** 2021

Projektbezeichnung: Einrichtung einer verteilten Monitoringlösung auf Basis von Prometheus und M3 im Bereich des Gesundheitswesens.

Projektbeginn: 26.02.2021 Projektfertigstellung: 28.04.2021 Zeitaufwand in Stunden: 35

Bestätigung der Ausbildungsfirma:

Wir bestätigen, dass der/die Auszubildende das oben bezeichnete Projekt einschließlich der Dokumentation selbständig ausgeführt hat.

netzlink
Informationstechnik GmbH
IT-Campus Westbahnhof
WESTBAHNHOF 11
38118 BRAUNSCHWEIG
TELEFON 0531 - 707 34 30

28.04.2021 — N. Kahlert

Datum — Stempel der Firma — Unterschrift

Die Dokumentation darf zu Unterrichtszwecken an der Berufsschule verwendet werden. ○ JA ☒ NEIN

Eidesstattliche Erklärung:

Ich versichere, dass ich das Projekt und die dazugehörige Dokumentation selbständig erstellt habe.

Braunschweig, 23.04.2020 — Kahlert

Ort und Datum — Unterschrift des Prüflings

Prüfungsteil A: Betriebliche Projektarbeit