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

Nico Kahlert

**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 ).

# Einleitung

Im Zuge der andauernden wurde der Auftrag zum Betrieb der in den Rechenzentren des an die erteilt. Aufgrund eines raschen Anstiegs von zu überwachenden Maschinen in unserer Infrastruktur im Rechenzentrum des 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 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.

## Vorstellung des Kunden

Das 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 . Dem , 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 .

## Projekteinordnung

Da es sich beim Betrieb von 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 , nach welchem der Service von strukturiert ist, ist das Monitoring eine wichtige Datenquelle für das und somit ein Indikator für die Zufriedenheit des Kunden .

## 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.

## Verwendete Software

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

### Sormas

ist eine Software zum Pandemiemanagement und wurde vom aus Braunschweig zusammen mit der Firma Vitagroup für die Ebola-Epidemie von Ostafrika 2014 entwickelt. In Folge der wird diese Webanwendung für den behördlichen Gesundheitsdienst im Auftrag des von im Rechenzentrum des 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 in -Containern betrieben, um ein möglichst leichtes Lifecyclemanagement und problemlose zu garantieren. Aktuell wird je Gesundheitsamt eine -Instanz inklusive Datenbank und Proxy auf einer virtuellen Maschine ausgerollt.

### 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 basierte Datenformat in Textform (siehe Abbildung [figure:metrics] auf Seite ). In einem typischen szenario werden diese Daten, welche auch als Metriken bezeichnet werden, über einen -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 datenbanken mittels des *Prometheusremoteprotocols* zu nutzen. Die Metriken können über die API oder der Weboberfläche abgegriffen werden. Zum Einholen der Metriken wird die verwendet (siehe Abbildung [figure:promql] auf Seite ). Innerhalb von containerbasierten infrastrukturen gilt Prometheus im Verbund mit Grafana als Standard.

### Nodexporter

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

### Grafana

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

### M3DB

M3DB ist eine quelloffene verteilte datenbank, welche bei dem Onlinevermittlungsunternehmen für Personenbeförderung von Privatpersonen Uber als Alternative zur Datenbank entwickelt wurde . Sie ist in der Programmiersprache verfasst und basiert auf einem Modell. M3DB ist nach dem als eine *CP*-Datenbank einzuordnen, da sie eine Ausfalltoleranz mit dem Replikationfaktor (Wobei N die Anzahl der Knoten ist) arbeitet und verschiedene garantierte Konsistenzlevel anbietet. Im Hintergrund verwendet die Datenbank ein externes oder eingebettetes -Cluster, in welchem die Konfiguration von *Clustermembership*, und Platzierung von stattfindet. Die Datenbank implementiert außerdem die Funktion eines *Prometheusremoteprotokollendpunketes* zur Benutzung als datenbank für Prometheus. Das Management der Datenbank ist über eine -API möglich. Die Daten werden von der M3DB in Dateien auf dem Filesystem als sogenannte *Filesets* im eigenen M3TSZ und format 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 .

### Consul

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

### 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 Änder- ungen implizit auf den Zielsystemen meist über eine einfache -Verbindung um, so dass oftmals keine weitere Software als auf den Zielhosts installiert sein muss. Die Funktionen von Ansible können über Module in erweitert werden. Die Informationen über den Zielzustand des Systems werden in einer oder mehreren Dateien im -Format, sogenannten , beschrieben (siehe Abbildung [figure:inv] auf Seite ). Um Wiederverwendbarkeit und eine gute Codeübersicht zu garantieren, können Teile eines in sogenannte gebündelt werden. Diese haben das gleiche Format wie , sind jedoch, im Gegenteil zu jenen, einer festen Ordnerstruktur unterworfen. Die von Ansible verwalteten Server werden in einem verwaltet, welches als Liste von Servern im - oder -Format vorliegt (siehe Abbildung [figure:ansible-inv] auf Seite ).

# Projektplanung

## 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 [table:aufwand] auf Seite )*.

## Projektschnittstellen

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

## Projektablaufplan

Das Projekt habe ich innerhalb der Regelarbeitszeit zwischen 8:00 Uhr und 18:00 Uhr umgesetzt. Der Projektablaufplan, *siehe Tabelle [table:gantt] auf Seite* , 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 visualisiert. Jedoch waren Überschneidungen, zum Beispiel die Bearbeitung von Supportanfragen im Betrieb der -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.

## Durchführung der IST-Analyse

Vor der Durchführung des Projektes bestand das Monitoring der Infrastruktur im 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 -Basis.  
Die minütige Datenmenge *D (MB/min)* des Prometheusservers konnte ich mit der Größe der Metriken *DM (KB)*, dem Scrapeintervall *t (s)* und der Anzahl der zu überwachenden Server *(n)* kalkulieren.

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 im Bereich des Monitorings und der daraus folgenden Alarmierung bei Ausfällen im Service.

## Ermittlung des SOLL-Zustandes

Auf drei virtuellen Maschinen mit derselben Konfiguration (siehe Punkt [ssec:ist]) richte ich ein dezentrales Monitoringsystem auf der Basis der Technologien Prometheus und der M3DB ein. Die Implementierung soll einfach *horizontal* skalierbar sein und keinen haben. Außerdem ist eine Anforderung, dass alle bisherigen Grafana Dashboards weiter genutzt werden können. Um nicht auf allen drei Prometheusinstanzen die Konfiguration über alle zu monitorenden -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 [figure:node] auf Seite ). Zur einfachen Wartbarkeit und Installation verwende ich die bereitgestellten der Produkte.  
Aus Gründen der einfachen Installation, Skalierbarkeit und Wartung verwende ich die in unserem Team oft eingesetzte Automatisierungssoftware Ansible.

## Evaluierung der VMs

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

# Projektrealisierung

## 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 bei der Entwicklerplattform *Github* eingerichtet. Um nicht immer wieder Zeit für multiple Authentifikation zu verlieren, habe ich mich hier für die -Option zur Anmeldung mit meinem *Public-Key* entschieden (siehe Abbildung [figure:git] auf Seite ). Wie im Punkt [sssec:ansible] beschrieben benötigt die Automatisierungssoftware *Ansible* eine -Datei, welche ich in einen Unterordner mit dem Namen *inventory* als gleichnamige -Datei platzierte. Da das Projekt aber als Vorlage für mehrere potentielle Installationen dienen soll, habe ich den Pfad der -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 erstellt. Der letzte Teil für die Automatisierung mit *Ansible* war das Erstellen des an sich. Dafür habe ich eine Datei mit dem Namen *main.yml* im Projektverzeichnis erstellt. In dieser Datei habe ich das Playbook im -Format beschrieben (siehe Abbildung [figure:playbook] auf Seite ). Das Playbook habe ich mit einer einzigen konzipiert, welche die Anforderung stellt, dass die Zielsysteme alle im Array des Unterpunktes *roles* vorhandenen eingenommen haben. Nun konnte ich das Playbook mit dem *Ansible*-Kommando ausführen (siehe Abbildung [figure:ansible-command] auf Seite ). Da das Kommando mit seinen Optionen kompliziert war und der Anforderung der Einfachheit nicht entsprochen hätte, habe ich mich entschieden, eine -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 file die -Datei als Vorraussetzung eingetragen, damit , beim Fehlen dieser, eine Warnung ausgibt und den Vorgang abbricht, ohne den *Ansible*prozess zu starten (siehe Abbildung [figure:make-ansible] auf Seite ).

## Anlegen von Ansible Rollen

### 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 [figure:galaxy] auf Seite ). 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 für den Betrieb genutzt werden, herunterladen zu können. Als Nächstes habe ich den für die Installation der weiteren Pakete konzipiert (siehe Abbildung [figure:base-install] auf Seite ). Der 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 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 [figure:base-packages] auf Seite ). 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.

### Nodeexporter

Um die VMs des Systems selbst überwachen zu können, habe ich den Nodeexporter aus Punkt [sssec:nodexporter] auf Seite in das Playbook integriert. Ansible bietet der Interessengemeinschaft *(Community)* mit der *Ansible Galaxy* ein Repository zum Austausch von 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 [figure:requirements] auf Seite ). Diese 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.

### 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 (siehe Abbildung [figure:etcd-command] auf Seite ). 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 [figure:etcd-var] auf Seite ). 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 [figure:member-list] auf Seite ) 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.

### 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 [sssec:m3db] auf Seite beschrieben, verwendet. Das andere dient zum *Caching (Zwischenspeicherung)* der *Namespace*-Konfiguration und des *Placing* der im ETCD. Zum Erstellen dieser habe ich das file-Modul genutzt. Ein weiteres Verzeichnis enthält eine Konfigurationsdatei im -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 . 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 [figure:m3db-ports] auf Seite ). Nachdem das Cluster aufgebaut war, sind zur Nutzung mit Prometheus einige Schritte in der Datenbanklogik zu konfigurieren. M3DB bietet für diese Konfiguration eine -Schnittstelle an. Zuerst habe ich das uri-Modul verwendet, um einen Namespace mit dem Namen *default* im M3DB-Cluster, per -Payload zu initialisieren. Die nächste Konfiguration, die ich getätigt habe, ist die der Platzierung der s auf die Knoten des Clusters. Dazu habe ich die Anzahl der s auf 32 und den Replikationfaktor, aufgrund der Konfigurationsempfehlung der M3DB-Entwickler\*innen zur Produktion, auf drei gesetzt (siehe Abbildung [figure:m3db-placing] auf Seite ). Als letzten Schritt der Rolle muss ich Ansible angeben, auf die Fertigstellung des *Bootstrap*-Prozesses zu warten, bis dem uri-Modul am -Endpoint /api/v1/services/m3db/placement ein -Body übermittelt wird, in dem alle s den Status AVAILABLE haben. So wird sichergestellt, dass vorher keine anderen Systeme auf eine unvorbereitete Datenbank zugreifen.

### 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 [sssec:etcd-rolle] auf Seite zum ETCD beschrieben.

### Prometheus

Auf jedem Host wird ein eigenständiger Prometheusserver installiert, welcher die Metriken der 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 [figure:prometheus-conf] auf Seite ). 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.

### 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.

## Installation der Software

### Vorbereitung der Installation

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

### Durchführen der Installation

Zur Durchführung der Installation auf den Hosts des habe ich den Ansibleprozess über make gestartet, was zur Ausgabe in der Abbildung [figure:ansible-run] auf Seite 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 [figure:ansible-recap] auf Seite ).

## 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 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 -Abfragen abrufen ließen.

## 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 [figure:m3-dash] auf Seite ). 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.

# 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 . 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 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 noch keine Infrastrukturautomatisierung in dieser Dimension umgesetzt. Die Nutzung der und die Einteilung in 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.

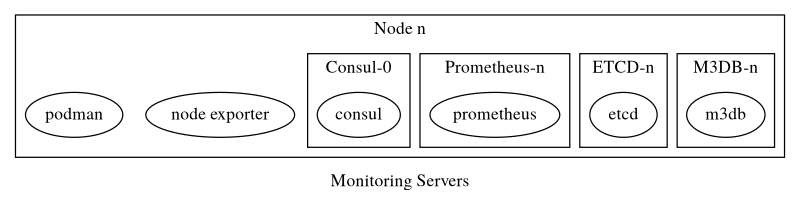


Diagramm zur Übersicht eines Monitoringknotens

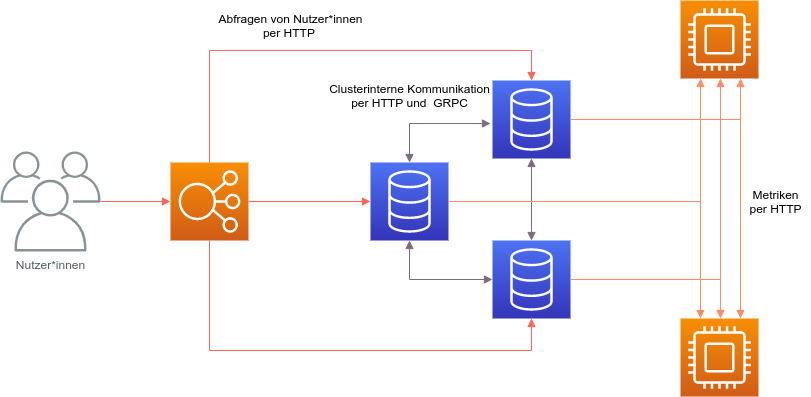
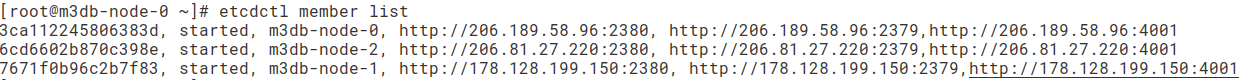
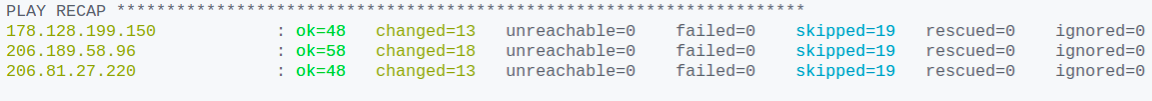


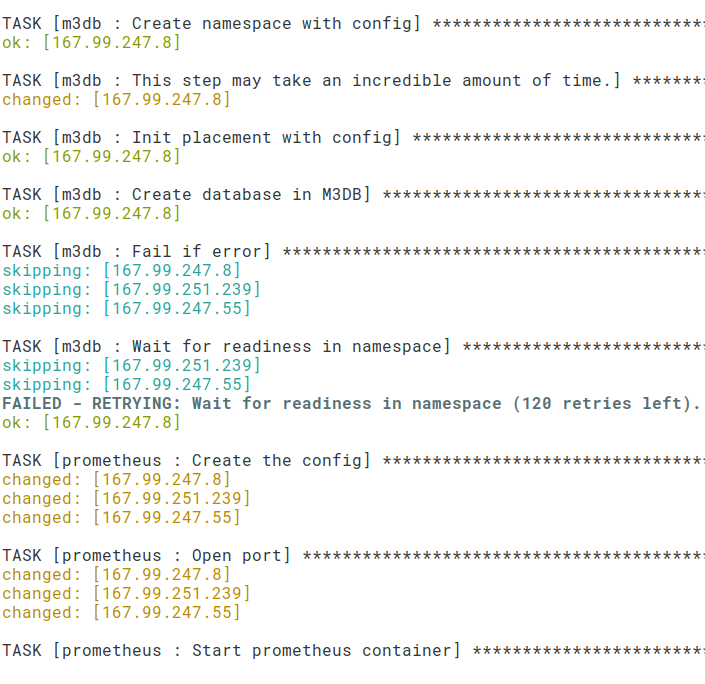
Diagramm zur Übersicht des Netzwerkes



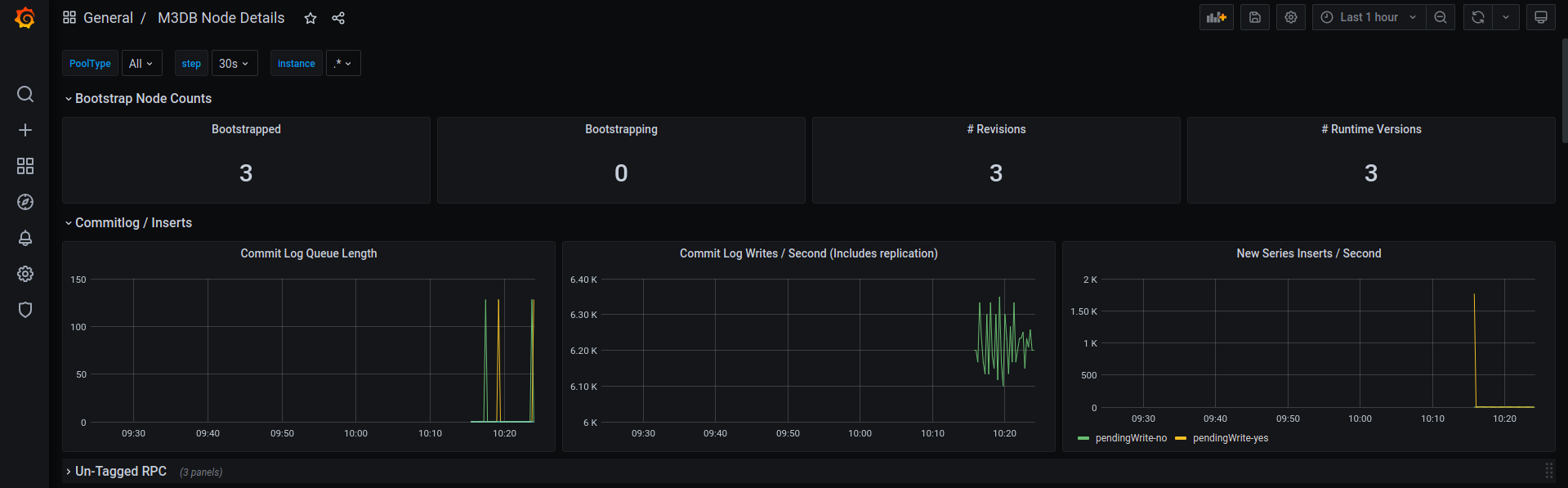
Ausgabe des ETCD zum Zustand des Cluster



Ausgabe der Zusammenfassung des Ansibleprozess



Ausgabe des Ansibleprozess



Grafana M3 Dashboard

[databse\_servers]  
 165.xxx.87.xxx  
 161.xxx.213.xxx  
 165.xxx.172.xxx

# 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

sum(rate(http\_requests\_total{job="prometheus"}[5d]))

---  
 - name: Postgres Playbook  
 hosts: database\_servers  
 become: yes  
 tasks:  
 - name: ensure postgres is installed  
 dnf:  
 name: postgresql-server  
 state: present

git init  
 git remote add origin git@github.com:nk-designz/distributed\_prometheus.git

ansible-galaxy init common

---  
 - name: Setup Distributed Prometheus  
 hosts: all  
 become: yes  
 remote\_user: root  
 roles:  
 - cloudalchemy.node\_exporter

ANSIBLE\_HOST\_KEY\_CHECKING=false \  
 ansible-playbook \  
 -e 'ansible\_python\_interpreter=/usr/bin/python3' \  
 -i inventory/hosts.ini \  
 main.yml

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

- name: Install common packages  
 dnf:  
 name: "{{ common\_packages }}"  
 state: present

# common packages for the nodes  
 common\_packages:  
 - vim  
 - htop  
 - podman  
 - firewalld  
 - python3-firewall  
 - jq

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

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

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"

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"

Aufwandsschätzung aller Kosten bezogen auf die Projektphasen

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

Aufwandsschätzung aller Kosten bezogen auf die Projektphasen

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

Ressourcen beim

GANTT-Diagramm der Projektplanung

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 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 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 |  |  |

GANTT-Diagramm der Projektplanung

|  |  |
| --- | --- |
|  | Projektplanung |
|  | Realisierung |
|  | Evaluation |
|  | Dokumentation |

9

https://www.bundesgesundheitsministerium.de/  
05.03.2021 8:43 Uhr

https://www.sormas-oegd.de/  
05.03.2021 8:45 Uhr

https://www.itzbund.de/  
05.03.2021 9:21 Uhr

https://etcd.io/  
09.03.2021 8:20 Uhr

https://raft.github.io/raft.pdf  
09.03.2021 8:33 Uhr

https://m3db.io  
09.03.2021 15:02 Uhr

https://etcd.io/docs/v3.4/op-guide/configuration/  
09.03.2021 9:40 Uhr

https://m3db.io/docs/operational\_guide/placement\_configuration/  
09.03.2021 16:01 Uhr

https://eng.uber.com/m3/  
09.04.2021 16:20 Uhr

https://www.youtube.com/watch?v=CcH13GyszHI  
09.04.2021 16:15 Uhr

https://cassandra.apache.org/  
05.03.2021 13:23 Uhr

https://de.wikipedia.org/wiki/Split\_Brain\_(Informatik)  
09.04.2021 15:32 Uhr

https://www.itzbund.de/DE/home/home\_node.html  
09.04.2021 12:08 Uhr

https://sre.google/sre-book/monitoring-distributed-systems/  
09.04.2021 7:24 Uhr

Introducing Go - Caleb Doxsey ISBN: 9781491941959  
2016

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

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

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

https://prometheus.io/docs/  
14.03.2021 15:08 Uhr

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

https://microservices.io/  
09.04.2021 14:17 Uhr

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

https://docs.ansible.com/ansible/latest/user\_guide/playbooks.html  
09.04.2021 10:01 Uhr

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

https://grafana.com/docs/  
09.04.2021 12:01 Uhr

https://podman.io/  
09.04.2021 10:01 Uhr

ITIL Foundation: ITIL 4 Edition - AXELOS ISBN: 9780113316144   
2019

Genehmigter Projektantrag  
Bestätigung über die durchgeführte Projektarbeit