Elasticsearch + Kibana + Integrations

Cloud Native Monitoring
FH Burgenland - Cloud Computing Engineering
Technisches Management von Cloud Solutions

Stephan Pilwax
2110781003@fh-burgenland.at

Thomas Stadler
2110781014@fh-burgenland.at

Klaus Thüringer
2110781007@fh-burgenland.at

***Abstract*—Ziel dieses Papers ist die Dokumentation und Beschreibung des Aufbaus einer Monitoringumgebung mittels Open Source Software. Mit diesen sollten jeweils Metriken, Traces und Logs überwacht werden und in einer grafischen Benutzeroberfläche dargestellt werden. Das Projekt wurde mittels Terraform auf AWS umgesetzt und der verwendete Code auf GitHub dokumentiert.**

***Keywords*—Elasticsearch, Kibana, Monitoring, Terraform, Metriken, Traces, Logs**

I. Einleitung

In cloudnativen Umgebungen gelten hohe Anforderungen an das Monitoring. Es müssen weitaus komplexere und vernetzte Infrastrukturen überwacht und observiert werden. Der Microservice-Ansatz im Software-Architektur-Design macht es mit steigender Anzahl an Diensten stetig schwieriger, eine Applikation ganzheitlich zu monitoren.

A. Logs

Logs werden statt auf einer oder einer Handvoll physischen Maschinen hingegen auf bis zu Hunderten verteilten Systemen geschrieben oder nicht geschrieben, sondern lediglich auf die Standardausgabe ausgegeben. Um diesen veränderten Rahmenbedingungen und Anforderungen nachzukommen, müssen Monitoring-Systeme angepasst werden. Cloudnative Monitoring-Tools aggregieren Logs einer Applikation von beliebig vielen vernetzten Systemen in eine gemeinsame Datenbank. Somit können diese Logs zum einen persistiert werden. Des Weiteren können die zusammengeführten Logs von einer Suchmaschine indiziert werden und im Anschluss effektiv durchsucht werden. Logs von verschiedenen oder verwandten Applikationen können miteinander verglichen werden. Mit Softwarelösungen wie Elasticsearch [1] können diese Daten analysiert werden, auch mithilfe von Machine Learning. Dies kann zum Beispiel eine Analyse auf Anomalien sein. Elasticsearch bietet noch viele weitere Funktionen im Bereich Analyse von Big Data. Logs geben in der Regel detailliertere Informationen über Ressourcen als Metriken.

B. Metriken

Metriken sind typischerweise als Zahlen dargestellte Daten, die meist in Relation zur relativen Zeit, welche in einer Time-Series-Datenbank persistiert werden. Ebenso sind unter Metriken jene Werte zu verstehen, die auch traditionelle Monitoring-Systeme wie beispielsweise Nagios [2] verwalten. Das sind Prozessor-Auslastung, Arbeitsspeicher-Auslastung, Festplattenkapazität, etc. Aber das vielleicht Wichtigste, was man über Metriken wissen muss, ist der Punkt über die Möglichkeit, Metriken über Infrastrukturkomponenten hinweg zu korrelieren. Angesichts der komplexen Abhängigkeiten, die heute in IT-Umgebungen üblich sind, ist die Fähigkeit, Metriken zu einem Gesamtbild zusammenzufügen, eine große Zeitersparnis. Mit dem Übergang zu cloudnativen Umgebungen wird dies aufgrund der dynamischen Natur der Cloud-Infrastruktur und der sich ständig ändernden Beziehungen zwischen dieser Infrastruktur und den Anwendungen noch wichtiger. Um also auftretende Fehler oder Unstimmigkeiten bei verteilten Anwendungen effizient erkennen zu können, kommen heutzutage in cloudnativen Monitoring-Systemen die sogenannten Traces zum Einsatz.

C. Traces

Traces sind, wie der Name schon verrät, Anwendungs-Trace-Daten, die Informationen über bestimmte Anwendungsvorgänge ”verfolgen”. Da heutzutage so viele Anwendungen voneinander abhängig sind, umfassen diese Vorgänge in der Regel Sprünge durch mehrere Dienste, sogenannte Spans. Traces bieten somit einen wichtigen Einblick in den Zustand einer Anwendung von Ende zu Ende. Sie konzentrieren sich jedoch ausschließlich auf die Anwendungsebene und bieten nur begrenzte Einblicke in den Zustand der zugrunde liegenden Infrastruktur. Selbst wenn Traces gesammelt werden also immer noch Metriken benötigt werden, um einen vollständigen Überblick über eine Umgebung zu erhalten. Application-Performance-Management-Tools (APM-Tools) speisen Trace-Informationen in einen zentralisierten Metrikspeicher ein, sodass Traces eine gute Datenquelle für eine anwendungszentrierte Sichtweise darstellen. Der Bedarf an der Sichtweise, die Traces bieten können, wird in containerbasierten Microservice-Architekturen, die nichts anderes als eine Sammlung von zusammengefügten Services sind, noch größer. Diese Umgebungen können mit dem sogenannten verteilten Tracing adressiert werden.

Verteiltes Tracing ist die dritte Säule der Observability und bietet Einblicke in die Leistung von Vorgängen über Microservices hinweg. Eine Anwendung kann von mehreren Diensten abhängen, von denen jeder seinen eigenen Satz an Metriken und Logs hat. Verteiltes Tracing ist eine Möglichkeit, Anfragen zu beobachten, während sie sich durch verteilte

Cloud-Umgebungen bewegen. In diesen komplexen Systemen zeigen Traces Probleme auf, die bei den Beziehungen zwischen einzelnen Diensten auftreten können.

D. Projektumsetzung

In folgenden dieses Papers wird die Installation, sowie die erste Einrichtung der open-source Lösungen Elasticsearch und Kibana beschrieben. Diese werden von dem amerikanischen Unternehmen Elasticsearch B.V. entwickelt. Diese Kombination der eigenständigen Tools Elasticsearch und Kibana wird auch ELK Stack genannt. Dieser Software-Stack kann mit weiteren von Elasticsearch B.V. entwickelten Software-Lösungen wie Logstash oder Beats zum Elastic Stack erweitert werden. Zusammen bildet der Stack ein mächtiges Tool zur cloudnativen Observability.

Observability bezeichnet dabei die Fähigkeit, den internen Zustand eines Systems zu verstehen, indem die von ihm erzeugten Daten wie Logs, Metriken und Traces analysiert werden.

II. Installation

Die im folgenden erläuterten Installationsschritte sind in der Datei `README.md` in dem Repository https://github.com/thomasstxyz/fhb-mcce-tmcsp-cloudnative-monitoring. genauer beschrieben.

A. Ausgangssituation und Vorraussetzungen

Als Basis-Betriebssystem wird Ubuntu Linux Server in Version 20.04 [3] verwendet. Worauf die Docker Engine [4] installiert ist.

Für den Betrieb von Elasticsearch als Container muss der Parameter `vm.max_map_count` mittels `sysctl` auf den Wert `262144` gesetzt werden.

```
sudo sysctl -w vm.max_map_count=262144
echo "vm.max_map_count=262144"|sudo tee \
--append /etc/sysctl.conf
```

Als Quelle der Container Images wird die Repository `docker.elastic.co` herangezogen. Es werden die Images `elasticsearch/elasticsearch:8.2.2`, sowie `kibana/kibana:8.2.2` verwendet. Hier ist anzumerken, dass alle Komponenten in einem Elastic Stack mit der selben Versionsnummer installiert werden müssen.

Die im folgenden installierten Container sollen alle im selben internen Docker-Netzwerk `elastic` laufen. Dieses wird mit dem Befehl `docker network create elastic` erstellt.

B. Start der Container

Nun wird die initiale Instanz von Elasticsearch gestartet, welche einen Elasticsearch-Cluster erstellt, mit anfangs noch einem Knoten. Dafür wird das `elasticsearch:8.2.2` Container Image mit `docker run` und einer zusätzlichen Umgebungsvariable `-e ES_JAVA_OPTS="-Xms1g -Xmx1g"`, gestartet. Dies erhöht den Heapspeicher, der für die Java Runtime zur Verfügung gestellt wird. Wird dies nicht gemacht, führt dies zu Schwierigkeiten beim Starten von mehreren Elasticsearch-Containern auf ein und derselben Maschine. Die Ports 9200 und 9300 werden zur Host-Maschine weitergereicht, sodass der Container auf diesen Ports auf der Host-Maschine erreichbar ist. An den `docker run` Befehl wird dazu `-p 9200:9200 -p 9300:9300` hinzugefügt. Mit `--net elastic` wird der Container dem zuvor erstellten Netzwerk zugeteilt, und mit `--name es01` wird der Name des Containers definiert. Dies führt zu folgendem Befehl:

```
docker run -d \
-e ES_JAVA_OPTS="-Xms1g -Xmx1g" \
--name es01 \
--net elastic \
-p 9200:9200 -p 9300:9300 \
-it \
docker.elastic.co/elasticsearch/elasticsearch:8.2.
```

Nach dem Absetzen des `docker run`-Befehls werden mehrere Informationen auf die Konsole ausgegeben. Diese sollten kopiert und an einem sicheren Ort abgespeichert werden. Dies sind:

- Passwort des elastic user
- Kibana enrollment token
- Elasticsearch enrollment token

Sollte das Password einmal zurückgesetzt werden müssen, kann dies der Befehl `bin/elasticsearch-reset-password` im Container durchführen.

Außderdem wird auf die Konsole der Befehl ausgegeben, der zum hinzufügen weiterer Elasticsearch-Cluster-Knoten notwendig ist. Wird dieser ebenfalls unter Docker betrieben, muss hierfür lediglich der Enrollment Token als `-e ENROLLMENT_TOKEN="<token>"` Umgebungsvariable beim `docker run`-Befehl angegeben werden.

Nun wird Kibana ebenfalls als Docker Container gestartet. Der Port 5601 wird zur Host-Maschine durchgereicht, auf diesem ist das Web Interface zu erreichen. Dies führt zu folgendem Befehl:

```
docker run -d \
--name kib-01 \
--net elastic \
-p 5601:5601 \
docker.elastic.co/kibana/kibana:8.2.2
```

Jetzt kann das Kibana Web Interface im Browser unter http://⟨instance_ip⟩:5601 geöffnet werden. Zunächst wird der Benutzer aufgefordert das Enrollment Token von Elasticsearch, und einen Security Code einzugeben, welcher auf der Konsole ausgegeben wird. Dann kann sich mit dem User `elastic` und dem generierten Passwort eingeloggt werden.

III. Weiterführendes Setup

Erfolgreich im Web Interface angelangt, ist es zu empfehlen, einen sogenannten Fleet Server aufzusetzen. Dieser ist für eine

einheitliche Verwaltung der Elastic Agents zuständig, also die Systeme, die in Folge überwacht werden sollen. Um den Fleet Server zu initialisieren, muss nur zu http://⟨instance_ip⟩:5601/app/fleet navigiert, und den Schritten gefolgt werden.

Schließlich können diverse sogenannte Integrationen [5] für Elasticsearch im Kibana Web Interface installiert werden. Integrationen für open-source Software wie Apache httpd oder NGINX, oder auch Integrationen in verschiedenste Cloud-Umgebungen und dessen Systeme wie Amazon S3, GitLab, oder Atlassians Jira.

IV. Logs

Im Reiter `Observability>Logs>Stream` können alle zusammengeführten Logs eingesehen und durchsucht werden (siehe Fig. 1).

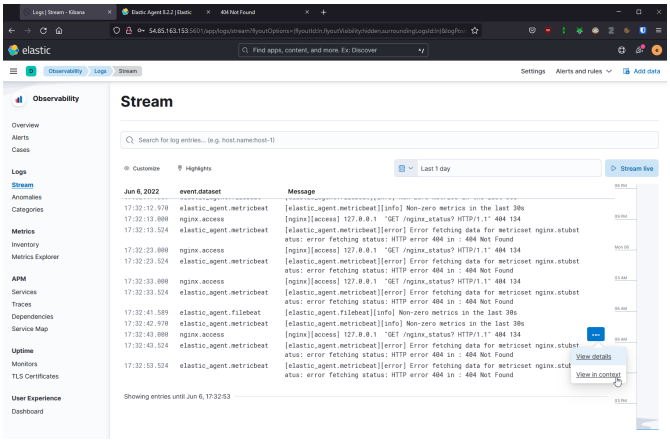

Fig. 1. Zusammenführung aller Logs als Stream

Nun kann beispielsweise die Ansicht auf die Nginx Logs eingeschränkt werden (siehe Fig. 2).

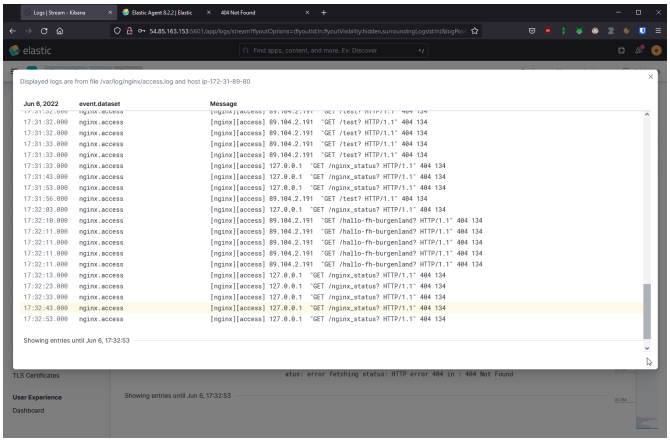

Fig. 2. Nginx Logs

V. Analyse und Visualisierung

In der `Visualize Library` kann eine visuelle Darstellung der Zugriffe auf Nginx angesehen werden (siehe Fig. 3), bezüglich der Quell-IP-Addresse, wird auf die Geolocation geschlossen. Somit können Besucher einer Website geografisch geortet bzw. statistische Auswertungen durchgeführt werden. Eine ähnliche Ansicht existiert auch als Heatmap.

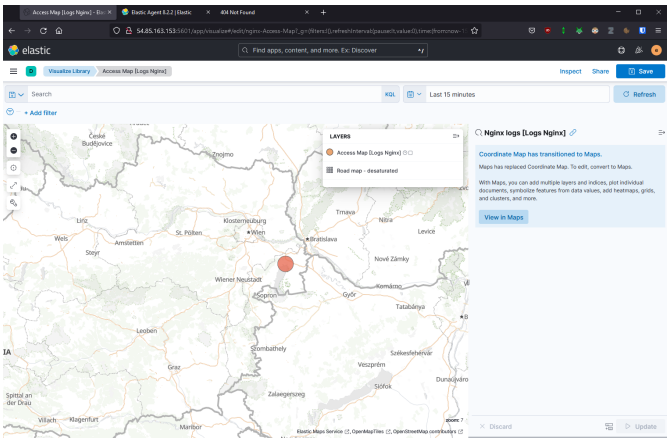

Fig. 3. Nginx Access Map

VI. Metriken

Im Reiter `Observability>Metrics>Inventory` können host-basierte Metriken wie Prozessor-Auslastung, Arbeitsspeicherverbrauch, Netzwerklast, und auch weitere Integrationen wie Metriken welche auf Basis der Nginx Logs gebildet werden, angesehen werden (siehe Fig. 4).

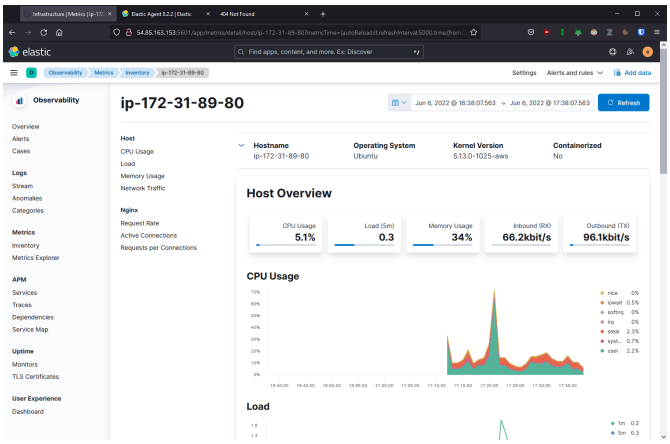

Fig. 4. Metriken

VII. Vergleich mit anderen Lösungen

Der ELK Stack (Elasticsearch, Kibana) kann zum Einlesen und Darstellen von Daten aus verschiedenen Quellen genutzt werden. Daten werden Dabei von Elasticsearch als unstrukturierte JSON Objekte gespeichert und sowohl die Keys als auch deren Inhalt indiziert. Dies ermöglicht es, die Daten später mittels JSON Objekt wieder abfragen zu können. Dazu können die beiden Query Sprachen Domain Specific Language (DSL) oder Lucene verwendet werden. Das Alternativ-Produkt zum ELK Stack von Grafana Labs, namens Grafana Loki [6]

speichert Daten im Single Binary Mode, welche im horizontal skalierten Modus in Cloud Speicher ausgelagert werden. [7]

Die protokollierten Daten werden im Klartext abgespeichert und für spätere Suche und Identifikation zusätzlich mit Labels versehen, welche dann für die Identifikation genutzt werden können. Im Gegensatz zu einer vollständigen Indexierung reduziert dieser Ansatz die Betriebskosten des Systems und bietet Entwicklern die Möglichkeit einer aggressiveren Protokollierung. Der große Nachteil dieses Ansatzes ist die Suche nach Text innerhalb des abgespeicherten Eintrags. Für diese müssen alle Chunks innerhalb des Suchfensters geladen werden. Diese können jedoch duch die zusätzliche Spezifizierung mittels eines Suchlabels für eine schnellere Suche eingeschränkt werden. Für das Suchen von aggregierten Logs für Loki bietet Grafana die Query Sprache LogQL an. [8]

Fluentd [9], welche nicht dem Unternehmen hinter Elasticsearch gehört, wird jedoch oft als zusätzliches Tool mit deren Produkten zur Datensammlung verwendet. Fluentd kann verwendet werden, um Logs aus vielen Quellen aufzunehmen, zu verarbeiten und an ein oder mehrere Ziele weiterzuleiten. Im Gegensatz dazu ist die Lösung von Grafana Labs, namens Promtail [10] auf Loki zugeschnitten. Die Hauptfunktion von Promtail ist es jedoch, gespeicherte Protokolldateien zu erkennen und sie in Verbindung mit einer Reihe von Labels an Loki weiterzuleiten. Promtail funktioniert ähnlich simpel wie Prometheus und exportiert offene Telemetriedaten in einfacher Textform. So kann Promtail beispielsweise Kubernetes Pods selbstständig erkennen, welche auf dem gleichen Node wie Promtail laufen oder als Container Sidecar genutzt werden. Es bietet ebenfalls Integrationen für Docker Container und kann Logs von ausgewählten Containern auslesen.

Die Protokollerstellung von Loki durch Label Paare ist sehr ähnlich zur Erstellung von Metriken von Prometheus. Wird Loki in einer Umgebung zusammen mit Prometheus eingesetzt, haben die Protokolle von Promtail in der Regel die gleichen Labels wie die Metriken der Anwendungen, da sie die gleichen Mechanismen zur Erkennung von Services nutzen. Die gleichen Bezeichnungen für Protokolle und Metriken, erleichtern es Benutzer nahtlos zwischen Metriken und Protokollen wechseln und vereinfacht den Analyseprozess. [8]

Für die Visuelle Aufbereitung von Elasticsearch Daten wird meist Kibana verwendet, welches ein sehr leistungsstarkes Tool für die Erstellung von Analysen der Daten ist. Kibana bietet eine Vielzahl an Visualisierungstools für die Datenanalyse, wie beispielsweise Standortkarten, maschinelles Lernen für die Erkennung von Anomalien und Diagramme zur Erkennung von Beziehungen zwischen Daten. Zusätzlich können in Kibana Warnmeldungen konfiguriert werden, um Benutzer zu benachrichtigen, wenn einer der vordefinierten Zustände auftritt.

Grafana hingegen ist speziell auf die Analyse von nach Zeit gereihten Daten aus Quellen wie Prometheus und Loki spezialisiert. Es können Dashboards eingerichtet werden, um Metriken zu visualisieren zusätzlich kann die Explorer Ansicht verwendet werden, um Abfragen für Daten zu erstellen. Wie auch Kibana unterstützt Grafana die Versendung von Alarmierungsnachrichten auf der Grundlage der vorliegenden Metriken. [8]

References

[1] Elasticsearch - the heart of the free and open elastic stack. https://www.elastic.co/elasticsearch/, 2022. (Accessed on 05/27/2022).

[2] Nagios - the industry standard in it infrastructure monitoring. https://www.nagios.org/, 2022. (Accessed on 05/27/2022).

[3] Ubuntu server. https://ubuntu.com/download/server, 2022. (Accessed on 05/27/2022).

[4] Docker engine. https://docs.docker.com/engine/, 2022. (Accessed on 05/27/2022).

[5] Elasticsearch integrations. https://www.elastic.co/integrations/, 2022. (Accessed on 05/27/2022).

[6] Grafana loki. https://grafana.com/oss/loki/, 2022. (Accessed on 06/18/2022).

[7] Grafana loki — grafana loki documentation. https://grafana.com/docs/loki/latest/. (Accessed on 06/17/2022).

[8] Comparisons — grafana loki documentation. https://grafana.com/docs/loki/latest/fundamentals/overview/comparisons/. (Accessed on 06/17/2022).

[9] Build your unified logging layer. https://www.fluentd.org/, 2022. (Accessed on 06/18/2022).

[10] Promtail. https://grafana.com/docs/loki/latest/clients/promtail/, 2022. (Accessed on 06/18/2022).