

# Fachhochschule Aachen Campus Köln



Fachbereich 9: Medizintechnik und Technomathematik  
Studiengang: Angewandte Mathematik und Informatik

## Logging, Tracing & Monitoring - Verfahren zur Überwachung von Anwendungen

**Bachelorarbeit**

von

**Natalie Fritzen**

Prüfer: Prof. Dr. rer. nat. Karola Merkel  
Zweitprüfer: B. Sc. Sebastian Otto  
Matrikelnummer: 3240219

Niederkassel-Rheidt, den 25. Juni 2022



# Sperrvermerk

Die nachfolgende Bachelorarbeit enthält vertrauliche Daten und Informationen der AXA Konzern AG. Veröffentlichung, Vervielfältigung oder die Weitergabe des Inhalts der Arbeit im Gesamten oder in Teilen sowie das Anfertigen von Kopien oder Abschriften (auch in digitaler Form) sind grundsätzlich untersagt. Ausnahmen bedürfen der schriftlichen Genehmigung der AXA Konzern AG. Die Bachelorarbeit ist nur den Korrektoren sowie den Mitgliedern des Prüfungsausschusses zugänglich zu machen.



# Eidesstattliche Erklärung

Ich versichere hiermit, dass ich die vorliegende Bachelorarbeit mit dem Thema

*Logging, Tracing & Monitoring - Verfahren zur Überwachung von  
Anwendungen*

selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, wobei ich alle wörtlichen und sinngemäßen Zitate als solche gekennzeichnet habe. Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Niederkassel-Rheidt, den 25. Juni 2022

---

Natalie Fritzen



# Abstract





# Inhaltsverzeichnis

<b>1</b>	<b>Einleitung</b>	<b>11</b>
<b>2</b>	<b>Definition Logging, Tracing &amp; Monitoring</b>	<b>12</b>
2.1	Logging . . . . .	12
2.2	Tracing . . . . .	16
2.3	Monitoring . . . . .	17
<b>3</b>	<b>Verfahren</b>	<b>19</b>
3.1	Standards aus verschiedenen Sprachen . . . . .	19
3.1.1	Java . . . . .	19
3.1.2	JavaScript . . . . .	21
3.1.3	Python . . . . .	21
3.1.4	C++ . . . . .	23
3.2	Logging . . . . .	23
3.3	Tracing . . . . .	24
3.4	Monitoring . . . . .	24
3.5	Weiterentwicklungen . . . . .	24
3.6	Performance . . . . .	24
<b>4</b>	<b>Nutzen im Projekt</b>	<b>25</b>
4.1	Aktuelle Nutzung . . . . .	25
4.2	Verbesserungen . . . . .	25
<b>5</b>	<b>Zusammenfassung und Ausblick</b>	<b>26</b>
	<b>Literatur</b>	<b>27</b>



# **1 Einleitung**

## 2 Definition Logging, Tracing & Monitoring

Es existiert kein fehlerfreies IT-System. Daher muss ein Prozess zu einem funktionierenden System geschaffen werden. Logging zeichnet auf, was eine Anwendung macht. Tracing protokolliert, wenn eine Anwendung andere Service aufruft. Monitoring zieht aus diesen Informationen Schlüsse, um Fehler einer Anwendung aufzudecken. Wenn die gefundenen Fehler ausgebessert werden, kommt man dem fehlerfreien System näher. Im Folgenden werden die drei Verfahren genauer erklärt.

### 2.1 Logging

Als Logging wird das automatische Protokollieren von Ereignissen bezeichnet. Hierzu zählen Fehlermeldungen, Systemnachrichten, Statusmeldungen, etc. Die Daten, die protokolliert werden, werden in eine oder mehrere Log-Dateien geschrieben. Diese Dateien enthalten die Ereignisse mit Zeitstempel, welche meist chronologisch geschrieben werden.

Außerdem werden die einzelnen Logs mit einem Loglevel versehen, womit der Entwickler das Protokoll besser auswerten kann. Es gibt sechs verschiedene Loglevel. Das schwerwiegendste Level heißt „Fatal“, hierbei spiegeln die Logs wider, dass die Applikation in einem katastrophalen Zustand ist und eingegriffen werden muss. Falls dieses Level auftritt, sollte sofort eingegriffen werden, um die Applikation wieder verfügbar zu machen. Das zweite Loglevel nennt sich „Error“, hierbei läuft die Applikation weiter, ist aber auf einen Fehler getroffen. Der Administrator sollte die Anwendung so früh wie möglich untersuchen und den Fehler beheben. Das nächste Level für Logs ist „Warning“. Die Applikation

ist in einem Zustand, der nicht üblich ist. Dieser sollte analysiert werden, um wieder in den Normalzustand zu gelangen. Da die Anwendung läuft, hat die Fehlerbehebung nicht die höchste Priorität, sollte trotzdem zeitnah erledigt werden. Als viertes Loglevel wird „Information“ bezeichnet. Dieses Level wird in der produktiven Umgebung genutzt, um Interessantes im Produktionssystem nachvollziehen zu können. In Testumgebungen wird das Loglevel „Debug“ genutzt, um weitere Informationen auswerten zu können. Das letzte Loglevel wird „Verbose“ genannt, hierbei werden alle Details einer Anwendung dokumentiert. Durch das gesetzte Loglevel können die Logs in einem Protokoll nach diesem Parameter gefiltert werden. Hierfür muss eine einheitliche Struktur der Logs gewährleistet sein.

Das Log-Management kümmert sich um die langfristige Speicherung, sodass die Daten über lange Zeit nachvollziehbar sind. Hierbei sollten die Logs an einer zentralen Stelle gespeichert werden, wie zum Beispiel einer Datenbank. Logs sollten bestimmten Standards entsprechen, um die Integration mit zentralisierten Protokollierungsdiensten zu erleichtern. Die Logs können kurzfristig auch in der Konsole ausgegeben werden, können dadurch aber nicht langfristig ausgewertet. Lokale Speicherung der Logs in Dateien erschweren das Auswerten.

Das Event-Management stellt Funktionen bereit, um diese Daten auswerten zu können. Zusammengefasst Log- und Event-Management sammeln die Log-Daten, speichern sie, sodass die Daten genutzt werden können, um Probleme zu identifizieren, Prozesse nachzuvollziehen, Systemleistung zu optimieren oder Cyberangriffe und andere Sicherheitsvorfälle zu erkennen und abzuwehren.

Logs helfen dabei Sicherheitsvorfälle zu identifizieren. Falls gegen Richtlinien verstoßen wird, fallen diese in den Logs durch die Überwachung auf. In den Protokollen werden Informationen festgehalten, um Probleme und ungewöhnliches Verhalten nachvollziehen zu können.

Manche Logs sind durch den Gesetzgeber verpflichtend, um Ereignisse nachverfolgen zu können. Transaktionen im Bankumfeld sind verpflichtend aufzuzeichnen, um die Nachvollziehbarkeit dieser Aktionen sicherzustellen. Im Gesundheitswesen ist Logging gesetzlich gefordert, um Compliance-Richtlinien

zu erfüllen. Unter anderem gilt die Datenschutzgrundverordnung, um die Daten richtig zu verarbeiten.

Durch künstliche Intelligenz oder maschinelles Lernen können große Datenmengen automatisiert organisiert und ausgewertet werden. Logging sollte innerhalb einer Anwendung einheitlich verwendet werden. Das Logging sollte in der gesamten Organisation konsistent verwendet werden, um die Ereignisse, die in den Protokollen abgelegt sind, mit Ereignissen verschiedener Systeme vergleichen und verwalten zu können.

Die geloggten Ereignisse können aus verschiedenen Quellen protokolliert sein. Falls es eine Client-Software ist, können die Aktionen auf dem Desktop oder mobilen Gerät aufgezeichnet werden. Bei der Nutzung einer Datenbank sollten diese Aufrufe protokolliert werden. Beim Auslesen, Verändern oder Löschen eines Datensatzes sollten diese als Ereignisse im Protokoll aufgenommen werden. Falls vertrauliche Daten abgelegt sind, müssen diese auch vertraulich in den Logdateien abgelegt sein. Der Vertraulichkeitsgrad muss beibehalten werden, um die Daten weiterhin zu schützen.

Um die Sicherheitsstandards, die Alarmierung und Berichterstattung zu erfüllen, müssen das Niveau und der Inhalt in der Anforderungsanalyse und Entwurfsphase entwickelt und festgehalten werden. Niveau und Inhalt sollte in einem angemessenen Verhältnis zu dem Informationsrisiko stehen. Hierfür gibt es keine einheitliche Checkliste, da jedes Unternehmen, Organisation und Anwendung verschieden sind.

Jeder Eintrag in einem Protokoll muss „wann, wo, wer und was“ aufzeichnen. Wann steht hierbei für den Zeitstempel des Ereignisses mit Datum und Uhrzeit in einem internationalen Format. Dieser kann vom Protokollierungszeitpunkt abweichen. Wo steht für die Kennung der Anwendung, beispielsweise der Name und die Version. Die Anwendungsadresse sollte auch protokolliert sein, hierzu zählt die IP-Adresse und Portnummer des Servers. Wer ist aufgeteilt in menschlicher und maschineller Benutzer. Zum einen die Quelladresse, beispielsweise die IP-Adresse des Benutzers oder die Mobiltelefonnummer. Zum anderen die Benutzeridentität, falls dieser authentifiziert ist, zum Beispiel der Benutzername oder die Lizenznummer. Was steht für die Schwere des

Ereignisses, also dem Loglevel, und die Beschreibung des Ereignisses. Weitere mögliche Aufzeichnungen sind HTTP-Statuscodes oder interne Einordnung. Manche Ereignisse dürfen nicht direkt in die Protokolle geschrieben werden. Sie müssen gesondert behandelt werden, entweder entfernt, maskiert, gehasht oder verschlüsselt werden. Daten wie Quellcode, sensible Informationen oder Authentifizierungskennwörter gehören zu diesen besonders zu behandelnden Ereignissen. Verschlüsselungsschlüssel und andere Geheimnisse müssen geschützt werden. Inhaberdaten von Bankkonten und Zahlungskarten müssen besonders geschützt gespeichert werden. Anders zu behandeln sind Daten wie Dateipfad, interne Netzwerkdaten oder nicht sensible personenbezogene Daten. Hierfür muss die Organisation selbst entscheiden, wie sensibel diese Daten protokolliert werden sollen.

Nach Möglichkeit sollten Fehler bei der Eingabevalidierung in den Logs gespeichert werden. Authentifizierungserfolge und -fehler sollten protokolliert werden. Zugriff sollte kontrolliert werden und Autorisierungsfehler sollten geloggt werden. Anwendungsfehler und Systemereignisse, beispielsweise wie Laufzeitfehler oder Verbindungsprobleme, sollten im Protokoll gespeichert werden. Um Anwendungen und verwandte Systeme zu überwachen, sollte jeder Start und Stop aufgezeichnet werden. Optional sollten weitere Ereignisse protokolliert werden, um die Sicherheit einer Anwendung zu erhöhen. Übermäßiger Gebrauch sollte dokumentiert sein. In den Protokollen sollte verdächtiges oder unerwartetes Verhalten einer Anwendung oder eines Systems erkennbar sein.

Die Protokolle mit den gesammelten Ereignissen müssen nach dem Speichern vor Missbrauch geschützt werden. Beispielsweise müssen Daten vor unbefugten Zugriff bewahrt werden. Manche Protokolle enthalten Daten, die der Konkurrenz dienen können. Geschäftliche Werte können Journalisten von Nutzen sein, um Schätzungen über Einnahmen zu machen.[1–3]

## 2.2 Tracing

Ein Trace ist die direkte Visualisierung eines Requests beim Durchlauf durch eine Anwendung oder eine komplette Anwendungslandschaft. Tracing macht es möglich, nachzuvollziehen, in welchen Zuständen ein Objekt zuvor war. Allerdings erhöht Tracing die Komplexität des Codes und ist daher besser geeignet für Microservicearchitekturen. Ein Trace zeichnet die verschiedenen Aufrufe auf, beispielsweise einen HTTP-Request oder einen Datenbankaufruf. Während der Entwicklung einer Anwendung können Zusatzinformationen zum Verarbeitungsprozess nützlich sein. Neben den üblichen Ausgaben einer Anwendung helfen die Zusatzinformationen beim Testen der Anwendung.

Bei Microservices hilft das Distributed Tracing und es bezeichnet die verteilte Rückverfolgung. Hierbei werden einzelne Prozesse getrennt und somit wird erkennbar, wo Probleme entstanden sind. Um bei Problemen den verursachenden Microservice zu identifizieren, erhält jeder Prozess eine Profil-ID. Jeder Microservice signiert jede Nutzungsanfrage mit dieser ID. Alle Prozesse lassen sich so einem bestimmten Service zuordnen, identifizieren und analysieren. Falls nun ein Fehler auftritt, lässt sich der Microservice, der den fehlerhaften Prozess ausführt, durch die ID identifizieren. Insgesamt ist die Fehlersuche vereinfacht. Des Weiteren unterstützt Tracing beim Debugging.

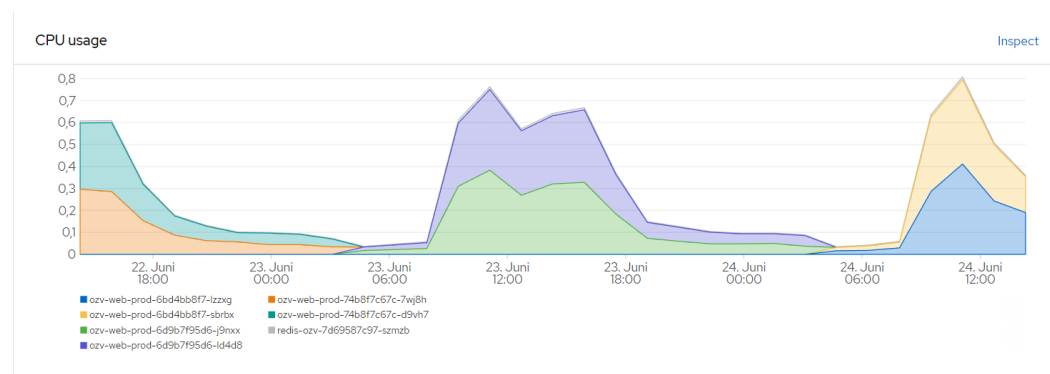
Tracing wird in verschiedenen Arten spezifiziert. Im allgemeinen Gebrauch für Anwendungen wird als Trace der Aufruf anderer Systeme bezeichnet. Während der Pandemie haben wir das ortsbezogene Tracing kennengelernt. Hierbei ist ein Trace die Nachverfolgung eines bestimmten Ereignisses, beispielsweise der Besuch einer Gastronomie. Außerdem gibt es das nähebezogene Tracing. Hierfür werden die Personen bzw. deren Geräte nachverfolgt und geschätzt wer, wie nah und wie lange in der Nähe war.

Oft wird Tracing mit Tracking verglichen oder sogar verwechselt. Der Unterschied liegt dabei, dass Tracking den aktuellen Zustand beschreibt, wie zum Beispiel die GPS-Koordinaten einer Bestellung. Tracing beschreibt das Nachverfolgen, für das gleiche Beispiel wird beim Tracing der Bestelleingang, Verarbeitungsprozess etc. protokolliert.[4–6]



## 2.3 Monitoring

Monitoring sammelt viele Daten und zielt darauf ab, richtige Schlüsse zu ziehen. Ein einfaches Beispiel, das Monitoring sollte erkennen, dass ein Problem vorliegt, wenn eine Komponente ausfällt. Hierfür wird die Anwendung konstant überwacht und der Status aller Komponenten erfasst. Des Weiteren werden die Daten der Anwendung oder Systems aufbereitet und bewertet, sodass sie in einer übersichtlichen Zusammenfassung präsentiert werden können. Durch Monitoring fallen Abweichungen vom Normalzustand auf, dadurch sollte ein Alarm ausgelöst werden, um den Nutzer aufzufordern den Fehler zu beheben. Je nach Schweregrad des Fehlers werden verschiedene Medien zum Benachrichtigen des Nutzers verwendet. Bei schweren Fehlern kann eine SMS auf das Arbeitstelefon geschickt werden. Bei leichteren Fehlern kann eine Email reichen.



**Abbildung 2.1:** CPU-Auslastung über 2 Tage

Durch den Vergleich zu historischen Daten soll das Monitoring-System Aussagen über die Zuverlässigkeit einer Anwendung treffen. Durch das Überwachen einer Anwendung können Ausfälle vermieden und vorgebeugt werden. Ein Interface des Monitorings zeigt die Performance und Auslastung der Komponenten, die durchgehend gemessen werden. Diese Überwachung wird genutzt, um Hardwareausstattung zu planen.

Anforderungen an ein Monitoring-System können in fünf Kategorien unterteilt werden. Zunächst beobachtet das Monitoring den „Zustand des Systems“.

Hierzu gehört das „End-to-End“-Monitoring, welches die ausgelieferten Daten so nah wie möglich am Endnutzer auf Funktionalität überprüft. Zur Zustandserfassung gehört die Statuserfassung der Dienste, unter anderem die Hardwareauslastung, als auch die Softwareüberwachung. Des Weiteren werden diese Informationen für lange Zeit gespeichert, um die Verfügbarkeit von Diensten und Komponenten zu analysieren. Die zweite Kategorie ist die „Alarmierung“, welche das manuelle Eingreifen verlangt. Hierbei wird ein Mitarbeiter über die Ursache des Fehlers informiert. Die Reaktionszeit und Fehlerbehebung wird dokumentiert. Die nächste Kategorie wird als „Diagnose“ bezeichnet, hierbei werden Informationen gesammelt, um die Ursachenanalyse von Fehlern detailliert zu ermöglichen. Die gesammelten Informationen dienen der Entscheidungsfindung. Die vierte Kategorie ist die „Qualitätsmessung“. Es werden Daten gesammelt, die über die Leistungsfähigkeit und den Durchschlag von Systemen und Anwendungen Aufschluss geben. Außerdem werden vereinbarte Grenzwerte und deren Einhaltung erfasst. Dadurch können Engpässe und Überlastungen aufgedeckt werden. Die letzte Kategorie ist die „Konfiguration“. Hierfür werden standardisierte Konfigurationen überwacht und bei Abweichungen von standardisierten Vorgehen gewarnt.

Monitoring bietet viele Vorteile. Unter Anderem werden die Abhängigkeiten zwischen Anwendungen abgebildet. Außerdem werden Trendanalysen genutzt, um den Einsatz von Rechenleistung und Ressourcen zu verbessern.

Monitoring ist in zwei Arten vorhanden. Einmal gibt es das „Historical Monitoring“, welches proaktives Arbeiten fordert. Hierbei muss der Admin vorausschauend das Handeln planen. Mit Langzeitstatistiken werden Kapazitäten geplant und können die Budgetplanung des Unternehmens unterstützen. Der Admin erkennt die Missstände, informiert die Betroffenen und entwirft Lösungsansätze. Die zweite Art des Monitoring ist das „Real-Time-Monitoring“. Hierbei werden die Server überwacht und bei Problemen direkt reagiert. Im Idealfall werden Fehler registriert und behoben, bevor der Nutzer diesen bemerkt.[7–9]

# 3 Verfahren

## 3.1 Standards aus verschiedenen Sprachen

### 3.1.1 Java

Die Standardbibliothek bietet seit Java-Version 1.4 ein Logging-Framework. Dieses Paket enthält unter anderem einen Logger. Dieser kann durch den Import „import java.util.logging.Logger“ genutzt werden. Um einen Logger zu nutzen, muss er auch initialisiert werden. Hierfür wird der Logger mit dem Klassennamen als Parameter, der nutzenden Klasse, erstellt oder der entsprechende Logger gefunden:

```
private static final Logger LOGGER =  
    Logger.getLogger(classname.class.getName());
```

Um eine Nachricht mit dem Logger zu erstellen, muss die Methode entsprechend zu dem Loglevel genutzt werden. Für eine informelle Nachricht wird die Methode `info()` genutzt, beispielsweise:

```
LOGGER.info („Loggername: " + LOGGER.getName());
```

Hierbei muss der Logger mindestens das Level INFO konfiguriert haben, welches als Standardebene eingestellt ist. Hierdurch entsteht folgender Log mit path als Variable für den Package- und Klassenname:

```
Jun 23, 2022 1:19:30 PM path main INFO: Loggername: path
```

Die Nachrichten können auch über eine allgemeinere Methode erstellt werden. Diese nennt sich `log()` und nimmt als Parameter das Loglevel und die Nachricht an. Sowie ein Objekt, welches protokolliert werden soll. Es können Objekt wie Exceptions gespeichert werden. Um diese Methode nutzen zu können, muss ein weiterer Import gesetzt sein „`import java.util.logging.Level`“. Denn es werden die statischen Einträge der Levelklasse genutzt.

```
LOGGER.log(Level.SEVERE, „Eine Exception ist aufgetreten“, ex)
```

Hierbei entsteht eine Nachricht in gleichem Format:

```
Jun 23, 2022 1:19:31 PM path main SEVERE: Eine Exception  
ist aufgetreten.
```

Vorher wurde bereits Log4J, welches im Verlauf erläutert wird, entwickelt, weshalb ein starker Vergleich gezogen wurde. Der Standard aus Java ist nicht kompatibel mit Log4J. Des Weiteren ist es nicht so leistungsstark wie Log4J.

Das Logging aus Java hat verschiedene Loglevel, die den Schwierigkeitsgrad des Logs zeigen. Diese heißen SEVERE, WARNING, INFO, CONFIG, FINE, FINER und FINEST. Je nach Loglevel werden die Nachrichten protokolliert oder ignoriert. Falls der Logger mit dem Loglevel WARNING konfiguriert wurde, wird die Nachricht mit Level INFO ignoriert. Daher kann unnötiger Aufwand entstehen, falls eine Nachricht, die aufwändig aufgebaut wird, nicht geschrieben wird, jedoch vorher schon eine Variable ausgewertet hat. Im diesen Aufwand zu verringern wurde die Methode `boolean isLoggable(Level level)` der Bibliothek hinzugefügt. Diese wertet den Ausdruck in der Log-Nachricht erst aus, falls die Nachricht geloggt werden darf. Eine weitere Möglichkeit den Aufwand zu verringern gibt es seit Java-Version 8. Hier wurden Lambda-Ausdrücke eingeführt, um den Programmcode in einem Block zusammen erst auszuführen, falls das Logging-Framework den Block ausführen soll.

Das Logging-Framework enthält als Komponente einen Handler, der die geloggen Nachrichten an einen bestimmten Zielort speichert. Der `KonsolenHandler`

schreibt die geloggten Nachrichten in die Konsole. Der FileHandler speichert die Logs in Dateien. Es gibt noch weitere Handler, die für das Speichern der Daten genutzt werden können.

Eine weitere Komponente des Logging-Frameworks ist ein Formatter. Jedem Handler ist ein Formatter zugeordnet. Es wird unterschieden zwischen SimpleFormatter und XMLFormatter. Der SimpleFormatter wird genutzt, um die Logs für den Menschen einfach lesbar zu machen. Für den FileHandler ist der XMLFormatter standardisiert. [10–12]

#### 3.1.2 JavaScript

Bei der Entwicklung einer Webanwendung können die Fehlermeldung in der Konsole ausgegeben werden. Bei veröffentlichtem Code ist die Konsole nicht verfügbar. Daher wird die Möglichkeit genutzt, die Fehlermeldungen bei dem Client zu sammeln und an einen Server zu senden, wo diese Logs auszuwerten sind. Auf der Konsole können mehrere Methoden verwendet werden, die den verschiedenen Logleveln entsprechen.

Für JavaScript gibt es vier Loglevel. Das Geringste ist für Klartext und nutzt die Methode „log()“. Das nächste Level ist Info mit der Methode „info()“. Außerdem gibt es wie bei anderen Log-Frameworks Warn und Error mit „warn()“ und „error()“. Mit diesen Aufrufen werden die Nachrichten entsprechend unterlegt, entsprechend dem Level. Hierfür werden die Methoden wie folgt aufgerufen:

```
console.log(msg) [13]
```

#### 3.1.3 Python

In der Standardbibliothek von Python ist ein Logging-Modul enthalten. Dieses kann über den Import „`import logging`“ genutzt werden und hat wie andere Loggingverfahren eine Logger, Handler, Filter und Formatter als Klassen in diesem Paket. Der Logger wird, wie in anderen Programmiersprachen, nicht direkt initialisiert, sondern über die Methode `getLogger(__name__)` den aktuellen Logger zurückgegeben. Der Logger wird direkt im Code verwendet und gibt die Informationen an den Handler weiter. Dieser sendet die Log-Nachrichten an

den Zielort, beispielsweise eine Datei. Die Klasse `Filter` bietet die Möglichkeit, zu filtern, ob ein Datensatz protokolliert werden soll. Der Formatter erzeugt das Layout der Logs.

Wie auch bei den anderen Log-Verfahren gibt es verschiedene Schwierigkeitsgrade mit entsprechender Methode, die auf dem `Logger`-Objekt aufgerufen wird.

Level	Methode	Nutzung
Critical	<code>critical(msg, args, kwargs)</code>	schwerwiegendes Problem liegt vor
Error	<code>error(msg, args, kwargs)</code>	ein Problem ist aufgetreten, sodass eine Funktion nicht ausgeführt werden kann
Warning	<code>warning(msg, args, kwargs)</code>	eine unerwartete Situation oder ein zukünftiges Problem
Info	<code>info(msg, args, kwargs)</code>	Anwendung läuft ordnungsgemäß
Debug	<code>debug(msg, args, kwargs)</code>	Problem diagnose
Notset	<code>log(level, msg, args, kwargs)</code>	

**Abbildung 3.1:** Loglevel des Python-Loggers

Hierbei enthält `msg` das Format der Zeichenfolge, die die Nachricht hat. `args` sind die Argumente, die in die Nachricht eingefügt werden. Der letzte Parameter `kwargs` kann verschiedene Schlagwörter enthalten:

- `exc_info`: falls nicht `false`, werden die Exception-Informationen den Logs hinzugefügt
- `stack_info`: default `false`, falls `true` wird Stack an Logs angefügt
- `stacklevel`: default 1, falls größer werden dementsprechend viele Stackframes übersprungen
- `extra`: bspw. ein Wörterbuch

Bei der Methode `log` wird das Level als statische Funktion angegeben. Im Code sieht die Logging-Methode beispielsweise wie folgt aus:

```
d = {'clientip': '193.168.0.1', 'user': 'fbloggs'}  
logger = logging.getLogger('tcpserver')  
logger.warning('Protocol problem: %s', 'connection reset',  
              extra=d)
```

Hierdurch entsteht beispielsweise folgende Nachricht, die als Log im Protokoll gespeichert werden kann:

```
2022-06-23 13:19:50,154 192.168.0.1 fbloggs Protocol  
problem: connection reset
```

[14, 15]

### 3.1.4 C++

In der Standardbibliothek der Programmiersprache C++ ist kein Logger integriert. Jedoch ist er leicht zu erstellen und sollte durch ein Interface abgekapselt werden. Für ein Interface werden die log-Methoden virtualisiert, je nachdem welche Loglevel genutzt werden sollen. Minimal sollten Info, Warn und Error angelegt sein. [16]

## 3.2 Logging

Bis die Standardbibliothek von Java um einen Logger ergänzt wurde, wurde bereits das Log4J entwickelt. Log4J ist ein Open Source Projekt. Es wird durch eine Apache Software License verbreitet. Log4J ist eine populäre API, weshalb sie in mehrere Sprachen übersetzt wurde. Beispielsweise wurde sie in C++ übersetzt, dort heißt die API Log4cxx.

Die zentralen Begriffe der API sind Logger, Level, Appender und Layout. Logger entspricht dem Zentrum des Interesse. Dieser kümmert sich um die Logs. Erstellt sie, falls diese dem richtigen Level entsprechen. Level ist das Loglevel. Der Schwierigkeitsgrad des Ereignisses. Falls der Logger das entsprechende Level des Ereignisses hat, soll die Nachricht in das Protokoll geschrieben werden. Der Appender kümmert sich um das Anhängen der Ausgabe. Falls die Ereignisse in Dateien geschrieben werden, sorgt der Appender dafür, dass

die vorher geschriebenen Ereignisse nicht geschrieben werden. Layout ist das Format, indem die Nachrichten festgehalten werden, beispielsweise HTML oder XML. Das Logging mit Log4J lässt sich über Java Anweisungen konfigurieren. Es kann aber auch eine Konfigurationsdatei verwendet werden. Mittlerweile ist diese Bibliothek veraltet und dient nur als Grundlage für moderne Nachfolger.

Eine Verbesserung dieses Logging-Frameworks ist Log4J2. Die größte Verbesserung ist das asynchrone Protokollieren. Hierdurch wird das Multithreading möglich und der Logger bleibt in keiner Deadlock, wie es in Log4J1 war. Bei eigenständigen Anwendungen ist Log4J2 „müllfrei“, weshalb der Garbage Collector den Logger nicht entfernt. Durch das Plugin-System ist Log4J2 einfach zu erweitern. Beispielsweise einen neuen Appender hinzuzufügen ist einfach, da Log4J nicht verändert werden muss. Log4J2 unterstützt für den Appender gewünschte Formate. Zum Beispiel Logback kann nur ein bestimmtes Format mit dem Appender transportieren. [10, 11, 17]

## **3.3 Tracing**

## **3.4 Monitoring**

## **3.5 Weiterentwicklungen**

## **3.6 Performance**



## **4 Nutzen im Projekt**

### **4.1 Aktuelle Nutzung**

### **4.2 Verbesserungen**

## **5 Zusammenfassung und Ausblick**

# Literatur

- [1] S. Luber und A. Donner. *Was ist Logging/(Event-)Log-Management?* 23. Nov. 2021. URL: <https://www.ip-insider.de/was-ist-logging-event-log-management-a-1074430/> (besucht am 09.06.2022).
- [2] F. Bader. *Fehlerhandling und Logging - Wie macht man das eigentlich richtig?* 13. Sep. 2019. URL: <https://www.aitgmbh.de/blog/entwicklung/fehlerhandling-und-logging-wie-macht-man-das-eigentlich-richtig/> (besucht am 09.06.2022).
- [3] *Logging Cheat Sheet*. URL: [https://cheatsheetseries.owasp.org/cheatsheets/Logging\\_Cheat\\_Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/Logging_Cheat_Sheet.html) (besucht am 09.06.2022).
- [4] C. Kappel. *Logging vs Tracing*. 19. Mai 2022. URL: <https://www.adoo.de/de/news/blog/logging-vs-tracing-2.jsp> (besucht am 09.06.2022).
- [5] M. Kahl. *Tracing vs. Tracking – Mikro vs. Makro: Digitale Corona Lösungen*. URL: <https://monstar-lab.com/de/expertinsights/digitale-corona-loesungen/> (besucht am 09.06.2022).
- [6] S. Augsten. *Was ist Distributed Tracing?* 30. Apr. 2021. URL: <https://www.dev-insider.de/was-ist-distributed-tracing-a-1010389/> (besucht am 17.06.2022).
- [7] *Was ist Monitoring?* 20. Nov. 2020. URL: <https://www.cloudradar.io/blog/was-ist-monitoring> (besucht am 09.06.2022).
- [8] *Professionelles IT-Monitoring für einen reibungslosen IT-Betrieb*. URL: <https://www.wbs-it.de/loesungen/monitoring> (besucht am 09.06.2022).

- [9] S. Collet. *Monitoring: Definition und was IT-Monitoring leistet*. URL: <https://www.crossmedia-it.com/it-monitoring-managed-service-provider/> (besucht am 09.06.2022).
- [10] C. Ullenboom. *Java SE 8 Standard-Bibliothek*. (Besucht am 21.06.2022).
- [11] Baeldung. *Introduction to Java Logging*. 24. Okt. 2021. URL: <https://www.baeldung.com/java-logging-intro> (besucht am 21.06.2022).
- [12] R. Joshi. *java.util.logging Example*. 11. Juni 2014. URL: <https://examples.javacodegeeks.com/core-java/util/logging/java-util-logging-example/> (besucht am 22.06.2022).
- [13] G. Root. *JavaScript Logging Basic Tips*. 1. März 2019. URL: <https://stackify.com/javascript-logging-basic-tips/> (besucht am 23.06.2022).
- [14] *Logging facility for Python*. 25. Juni 2022. URL: <https://docs.python.org/3/library/logging.html> (besucht am 25.06.2022).
- [15] *Python-Logging: So machen Sie mit dem Python-Logging-Modul Skriptfehler ausfindig*. 12. Mai 2022. URL: <https://www.ionos.de/digitalguide/websites/web-entwicklung/python-logging/> (besucht am 25.06.2022).
- [16] W. Ziegelwanger. *Professionelles Loggen in C++*. 14. Dez. 2019. URL: <https://developer-blog.net/professionelles-loggen-unter-c/> (besucht am 25.06.2022).
- [17] *Welcome to Log4J2!* 23. Feb. 2022. URL: <https://logging.apache.org/log4j/2.x/manual/index.html> (besucht am 21.06.2022).