Notes zu Quellen

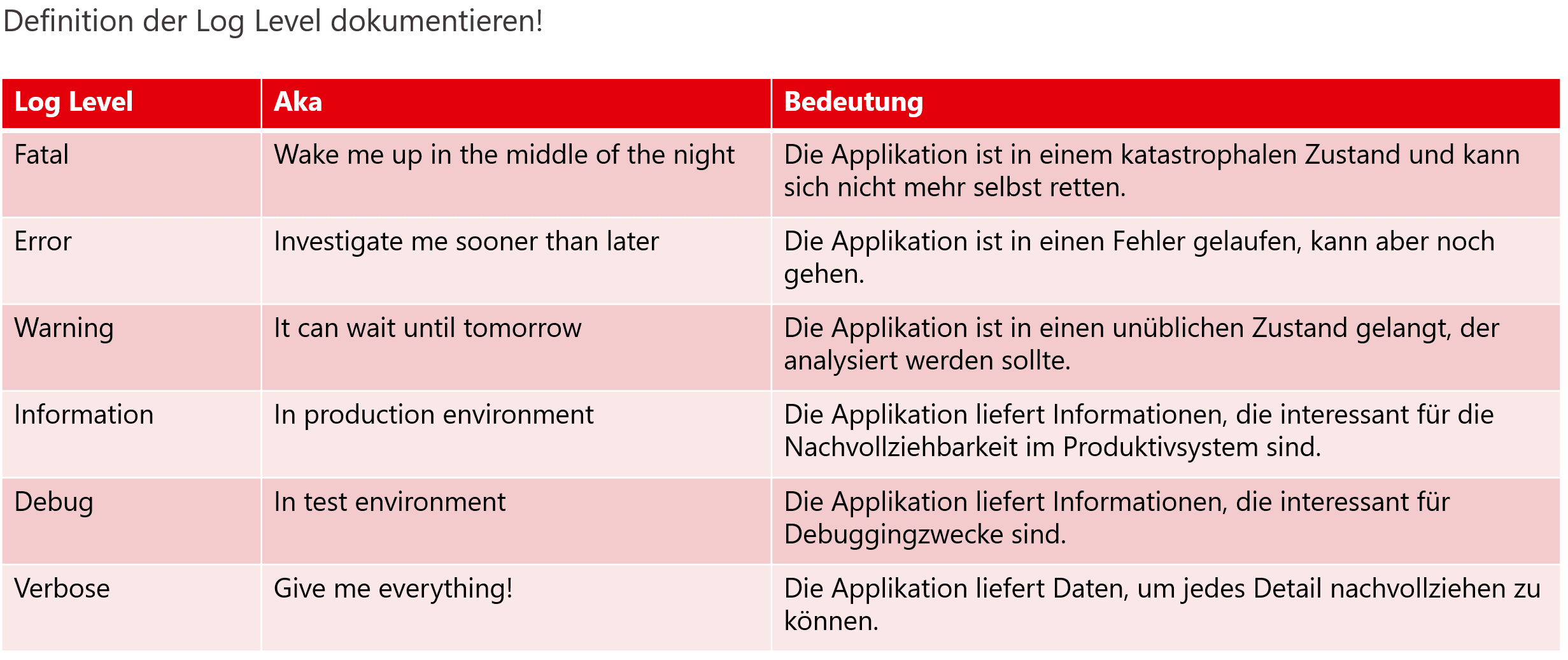
# Logging:

## Quelle: ip-insider

* Automatisches Protokollieren von Statusinformationen und Ereignissen
* Während Betrieb oder Ausführung von Systemen und Prozessen
* Fehlermeldungen, Systemnachrichten, Statusmeldungen, etc.
* In Logdateien sind Ergebnisse mit Zeitstempel und i.d.R. chronologisch
* Meist in Größe beschränkt, sodass nach gewisser Zeit überschrieben wird
* Log- und Event-Management hat die Aufgabe, die Log-Daten zu sammeln, aggregieren und speichern, sodass die Daten genutzt werden können, um Probleme zu identifizieren, Prozesse nachzuvollziehen, Systemleistung zu optimieren oder Cyberangriffe und andere Sicherheitsvorfälle zu erkennen und abzuwehren
* Teilweise gesetzlich verpflichtend Logs zu schreiben, bspw. Nachvollziehbarkeit von Transaktionen
* Sammeln der Logdaten erfolgt aktiv oder passiv: entweder Systeme senden Daten an Management oder Management holt sich Daten bei Systemen ab
* Standards und Protokolle: Syslog, SNMP (Simple Network Management Protocol), FTP (File Transfer Protocol) oder WMI (Windows Management Instrumentation)
* Aggregieren der Daten an zentraler Stelle, bspw. Datenbank
* Log-Management kümmert sich um langfristige Speicherung und Archivierung
* Event-Management stellt Funktionen zur Verfügung, zum Auswerten der Daten, bspw. Datenanalyse oder -suche
* Gesetzliche Anforderungen:
  + Finanzumfeld oder Gesundheitswesen ist Logging verpflichtend, um Compliance-Richtlinien zu erfüllen oder Nachvollziehbarkeit von Transaktionen sicherzustellen
  + Event-Management unterstützt: HIPAA (Health Insurance Portability and Accountability), SOX (Sarbanes-Oxley Act) oder DSGVO (Datenschutz-Grundverordnung)
* Softwarelösungen des Log-Management enthalten Komponenten zur Benutzeroberfläche für Analyse, Reporting und Visualisierung der Daten
* Durch Künstliche Intelligenz (KI) und Maschinelles Lernen (ML) können große Datenmengen automatisiert organisiert und intelligent ausgewertet werden
* SIEM (Security Information and Event Management): Lösung mit der sich sicherheitsrelevante Ereignisse, Sicherheitsvorfälle oder Anomalien innerhalb von IT-Umgebungen erkennen, beobachten und automatische Alarmierung und Abwehr ergreifen lassen
* SIEM-Lösungen liefern Überblick über die Sicherheitslage der IT und gewähren tiefe Einblicke in Aktivitäten der Systeme
* Vorteile:
  + liefert tiefe Einblicke in die Prozesse, Abläufe und in den Systemstatus der IT-Systeme und Netzwerke
  + verbessert die Observability der IT
  + stellt Funktionen bereit, mit denen sich Logdaten schnell durchsuchen lassen
  + ermöglicht umfangreiche Analysen der Logdaten
  + ermöglicht die Berichterstattung
  + hilft gesetzliche Vorgaben oder Compliance-Richtlinien einzuhalten
  + unterstützt beim Erkennen, Analysieren und Beheben von IT-Problemen
  + steigert die Sicherheit durch die Möglichkeit des Identifizierens, Nachvollziehens und Analysierens von Sicherheitsvorfällen
  + bildet die Basis für SIEM-Lösungen
  + verbessert die Reaktionsfähigkeit eines Unternehmens
  + stellt die Nachvollziehbarkeit von Transaktionen sicher
  + ermöglicht historische Analysen

## Quelle: ait gmbh

* in der Praxis bewährte Log-Anwendungen: Serilog, NLog und Log4net
* Protokollnachrichten sollten strukturiert sein, sodass in großen Protokollen besser nach Parametern filtern zu können
* Loglevel: wie dringend bzw wichtig ist eine Nachricht



* Wohin sollen Protokolle gespeichert werden
  + Console, Dateien, Event Log oder Datenbank
* Lokale Protokolle schwer auszuwerten

## Quelle: owasp

* Logging sollte innerhalb einer Anwendung und im gesamten Portfolio einer Organisation einheitlich und konsistent sein und ggf. Industriestandards verwenden, um die protokollierten Ereignisse von verschiedenen Systemen vergleichbar und verwaltbar zu machen
* Application Logs
  + Identifizieren von Sicherheitsvorfällen
  + Überwachung von Richtlinienverstößen
  + Grundlagen festlegen
  + Unterstützung von Unbestreitbarkeitskontrollen (beachten Sie, dass die Eigenschaft Unbestreitbarkeit für Protokolle schwer zu erreichen ist, da ihre Vertrauenswürdigkeit oft nur darauf basiert, dass die Protokollpartei ordnungsgemäß geprüft wird, während Mechanismen wie digitale Signaturen hier schwer zu verwenden sind)
  + Bereitstellung von Informationen über Probleme und ungewöhnliche Bedingungen
  + Beitragen zusätzlicher anwendungsspezifischer Daten zur Untersuchung von Vorfällen, die in anderen Protokollquellen fehlen
  + Schutz vor der Identifizierung und Ausnutzung von Schwachstellen durch Angriffserkennung
* Andere Arten von Ereignissen:
  + Sicherheitsereignisse
  + Überwachung von Geschäftsprozessen, z. B. Abbruch von Verkaufsprozessen, Transaktionen, Verbindungen
  + Anti-Automatisierungsüberwachung
  + Audit-Trails, z. B. Hinzufügen, Ändern und Löschen von Daten, Datenexport
  + Leistungsüberwachung, zB Datenladezeit, Seiten-Timeouts
  + Compliance-Überwachung
  + Daten für spätere Auskunftsanfragen, z. B. Auskunft über betroffene Personen, Informationsfreiheit, Rechtsstreitigkeiten, polizeiliche und andere behördliche Ermittlungen
  + Gesetzlich sanktioniertes Abfangen von Daten, z. B. Abhören auf Anwendungsebene
  + Andere geschäftsspezifische Anforderungen
* Bspw PCIDSS-Auditprotokolle enthalten chronologische Aufzeichnungen von Aktivitäten
  + Ermöglicht Rekonstruktion, Überprüfung und Untersuchung
* Ereignisdatenquelle ist die Anwendung, diese hat Informationen über den Benutzer und den Kontext des Ereignisses -> diese Daten sind meist nur innerhalb der Anwendung verfügbar und nicht in verwandten Anwendungen
* Andere Datenquellen
  + Client-Software, zB Aktionen auf Desktop oder mobilen Geräten
  + Netzwerk-Firewalls
  + Application Firewalls
  + Datenbankanwendungen, zB automatische Audit-Trails
  + Reputationsüberwachungsdienste, zB Betriebszeit- oder Malware-Überwachung
  + Andere Anwendungen, zB Betrugsüberwachung, CRM
* Vertrauensgrad muss berücksichtigt werden, wenn Ereignisdaten in anderen Vertrauenszonen einbezogen werden
* Zentralisiertes Protokollerfassungs- und Verwaltungssystem: zB SIEM oder SEM
* Standardformate über sichere Protokolle, um Ereignisdaten oder Protokolldaten aufzuzeichnen und an andere Systeme zu senden, zB Common Log File System(CLFS) oder Common Event Format(CEF)
  + Standards erleichtern die Integration mit zentralisierten Protokollierungsdiensten
* Niveau und Inhalt der Sicherheitsüberwachung, Alarmierung und Berichterstattung müssen währen Anforderungs- und Entwurfsphase entwickelt werden und sollten in einem angemessenen Verhältnis zu Informationsrisiken stehen
* Es gibt keine Einheitslösung, da es keine blinde Checkliste geben kann
* Nach Möglichkeit sollte:
  + Fehler bei der Eingabevalidierung, z. B. Protokollverletzungen, inakzeptable Kodierungen, ungültige Parameternamen und -werte
  + Fehler bei der Ausgabevalidierung, z. B. Nichtübereinstimmung der Datenbankdatensätze, ungültige Datenkodierung
  + Authentifizierungserfolge und -fehler
  + Autorisierungsfehler (Zugriffskontrolle).
  + Sitzungsverwaltungsfehler, z. B. Änderung des Cookie-Sitzungsidentifikationswerts
  + Anwendungsfehler und Systemereignisse, z. B. Syntax- und Laufzeitfehler, Verbindungsprobleme, Leistungsprobleme, Fehlermeldungen von Drittanbieterdiensten, Dateisystemfehler, Erkennung von Viren beim Hochladen von Dateien, Konfigurationsänderungen
  + Starten und Herunterfahren von Anwendungen und verwandten Systemen sowie Protokollierung der Initialisierung (Starten, Stoppen oder Anhalten)
  + Verwendung von Funktionen mit höherem Risiko, z. B. Netzwerkverbindungen, Hinzufügen oder Löschen von Benutzern, Änderung von Berechtigungen, Zuweisen von Benutzern zu Token, Hinzufügen oder Löschen von Token, Verwendung von Systemadministratorrechten, Zugriff durch Anwendungsadministratoren, alle Aktionen von Benutzern mit Administratorrechten, Zugriff an Zahlungskarteninhaberdaten, Verwendung von Datenverschlüsselungsschlüsseln, Schlüsseländerungen, Erstellung und Löschung von Objekten auf Systemebene, Datenimport und -export einschließlich bildschirmbasierter Berichte, Übermittlung von nutzergenerierten Inhalten - insbesondere Datei-Uploads
  + Rechtliche und andere Opt-Ins, z. B. Genehmigungen für Mobiltelefonfunktionen, Nutzungsbedingungen, Geschäftsbedingungen, Zustimmung zur Verwendung personenbezogener Daten, Erlaubnis zum Erhalt von Marketingmitteilungen
* Optional
  + Sequenzierungsfehler
  + Übermäßiger Gebrauch
  + Datenänderungen
  + Betrug und andere kriminelle Aktivitäten
  + Verdächtiges, inakzeptables oder unerwartetes Verhalten
  + Änderungen an der Konfiguration
  + Anwendungscodedatei und/oder Speicheränderungen
* Anwendungsprotokolle müssen für jeden Eintrag „wann, wo, wer und was“ aufzeichnen
* Wann:
  + Datum und Uhrzeit (internationales Format)
  + Zeitstempel des Ereignisses kann sich vom Zeitpunkt der Protokollierung unterscheiden
* Wo:
  + Anwendungskennung, zB Name und Version
  + Anwendungsadresse, zB Hostname oder Server-IP-Adresse und Portnummer, Workstation-Identität, lokale Gerätekennung
  + Dienst, zB Name und Protokoll
  + Geolokalisierung
  + Fenster/Formular/Seite, z. B. Einstiegspunkt-URL und HTTP-Methode für eine Webanwendung, Dialogfeldname
  + Ort des Codes, z. B. Skriptname, Modulname
* Wer (menschlicher oder maschineller Benutzer):
  + Quelladresse, z. B. Geräte-/Maschinenkennung des Benutzers, IP-Adresse des Benutzers, Zellen-/RF-Tower-ID, Mobiltelefonnummer
  + Benutzeridentität (sofern authentifiziert oder anderweitig bekannt), z. B. Primärschlüsselwert der Benutzerdatenbanktabelle, Benutzername, Lizenznummer
* Was:
  + Schwere des Ereignisses (LogLevel)
  + Flag für sicherheitsrelevante Ereignisse (wenn die Protokolle auch nicht sicherheitsrelevante Ereignisdaten enthalten)
  + Beschreibung
* Weitere Aufzeichnungen möglich:
  + Sekundäre Zeitquelle, zB GPS, Datum und Uhrzeit des Ereignisses
  + Ergebnisstatus – ob die auf das OBJEKT gerichtete AKTION erfolgreich war, zB Success, Fail, Defer
  + http-Statuscode – an den Nutzer zurückgegeben
  + interne Einordnung, zB Verantwortlichkeit, Compliance-Referenzen
* folgendes sollte nicht direkt protokolliert werden, sondern entfernt, maskiert, bereinigt, gehasht oder verschlüsselt
  + Quellcode der Anwendung
  + Sitzungsidentifikationswerte (erwägen Sie das Ersetzen durch einen Hash-Wert, falls erforderlich, um sitzungsspezifische Ereignisse zu verfolgen)
  + Zugriffstoken
  + Sensible personenbezogene Daten und einige Formen persönlich identifizierbarer Informationen (PII), z. B. Gesundheit, behördliche Kennungen, schutzbedürftige Personen
  + Authentifizierungskennwörter
  + Datenbank-Verbindungszeichenfolgen
  + Verschlüsselungsschlüssel und andere Hauptgeheimnisse
  + Bankkonto- oder Zahlungskarteninhaberdaten
  + Es dürfen Daten einer höheren Sicherheitsstufe gespeichert werden, als das Protokollierungssystem
  + Kommerziell sensible Informationen
  + Informationen, deren Sammlung in den entsprechenden Gerichtsbarkeiten illegal ist
  + Informationen, deren Erfassung sich ein Benutzer widersetzt hat oder deren Einwilligung er nicht zugestimmt hat, z. B. die Verwendung von „Do not track“, oder deren Einwilligung zur Erfassung abgelaufen ist
* Bevor ein Ereignis aufgezeichnet wir, sollten folgende Daten besonders behandelt werden
  + Dateipfade
  + Datenbank-Verbindungszeichenfolgen
  + Interne Netzwerknamen und -adressen
  + Nicht sensible personenbezogene Daten (z. B. Personennamen, Telefonnummern, E-Mail-Adressen)
* Protokollierungsmechanismen und gesammelte Ereignisdaten müssen nach der Speicherung vor Missbrauch geschützt werden (zB Manipulation bei der Übertragung, unbefugter Zugriff)
* Die Protokolle können geschäftliche Werte enthalten, wodurch sie für Konkurrenten, Journalisten etc. von Nutzen sein können, bspw Schätzung von Einnahmen

# Tracing:

## Quelle: adesso

* Trace: Direkte Visualisierung eines Requests beim Durchlauf durch eine Anwendung oder eine komplette Anwendungslandschaft
* Mit eindeutiger Trace ID und nimmt bei jedem Schritt Spans auf
* Spans: kleinste Einheit des Distributed Tracings und bilden Workflow ab, bspw. http Request, Datenbankaufruf
* Mit Span ID und Angaben über Timing, optionale weitere Attribute, Events, Status, je nach Use Case
* Logging:
  + Bietet Insights in einfache Anwendungen
  + Zeigt Status von Anwendungen
  + Kann einfach integriert werden
  + Gut geeignet für Monolithen
  + Unterstützt Debugging und Diagnosen
* Tracing:
  + Hilft, Requests durch Applikationen zu verfolgen
  + Liefert Timings und Latenzen für Requests
  + Erhöht die Komplexität des Codes
  + Besser geeignet für Microservicearchitekturen
  + Unterstützt Debugging und Diagnosen

## Quelle: Monstarlab

* Tracking: Echtzeit-Einblicke in den aktuellen Zustand eines Objekts (bspw GPS-Koordinaten des Lieferanten)
* Tracing: retrospektive Einblicke in die vorherigen Zustände eines Objektes (bspw. Nachverfolgung, wann was passiert ist: Bestelleingang, Bearbeitung, Verpackung und Lieferung)
* Ortsbezogenes (location-based) Tracing: Nachverfolgung, wenn ein bestimmtes Ereignis auftritt (bspw. Corona Listen in Gastronomien)
* Nähebeuogenes (proximity-based) Tracing: Nachverfolgung, wer in der Nähe ist (bspw. Corona Warn App: Schätzung über wer, wie nah und wie lange in der Nähe)

## Quelle: Dotnet

* Während Entwicklung, insbesondere Tests, neben üblichen Ausgaben der Anwendung, Zusatzinformationen zum Verarbeitungsprozess
* Verarbeitungsprozess verfolgen und kontrollieren
  + Ablaufverfolgung engl. Tracing
* Für spätere Auswertung verwenden
* Mittels Tracing einzelne Webformulare oder gesamte Anwendung

## Quelle: insider

* Distributed tracing: einzelne Prozesse in Microservices-Strukturen trennen und erkennbar machen
* Ziel: fehlerursachen schneller identifizieren
* Verteilte Rückverfolgung
* Um bei einem Problem, 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.
* Hilfreich für debugging

# Monitoring

## Quelle: cloudradar

* Es existiert kein fehlerfreies IT-System, daher muss der Prozess zu einem funktionierenden System durch Administratoren begleitet werden
* Monitoring hat folgende Aufgaben:
  + Status aller Komponenten erfassen
  + Daten aufbereiten, sortieren und bewerten
  + Übersichtliche Zusammenfassungen präsentieren
  + Abweichungen vom Normalzustand erkennen
  + Alarm auslösen
  + Zustände und Veränderungen protokollieren
  + Einhaltung von Prozessen oder eine Abweichung überwachen und protokollieren
* Monitoring sammelt viele Daten und zieht automatisiert richtige Schlüsse
* Fällt eine Komponente aus, als Schlussfolgerung: ein Problem liegt vor
* Je nach Schweregrad der Fehler, sollte das Monitoring-System verschiedene Medien zur Benachrichtigung nutzen
* Monitoring-System sollte Aussagen/Schlüsse über Zuverlässigkeit eines Systems/Komponenten geben
  + Nutzen historischer Daten
  + Monitoring-System sollte User-Interface zur Verfügung stellen, um das Auswerten zu vereinfachen
* Monitoring-System soll unterstützen, um Ausfall zu vermeiden/vorzubeugen (idR große Datenmengen notwendig)
* Performance und Auslastung der Komponenten muss permanent gemessen und dargestellt werden
* Monitoring-System sammelt Daten, um nicht vorgesehene Störfälle erklären zu können
  + Bspw. Zu hohe Besucheranzahl einer Webseite lässt den Server abstürzen
* Planung und Ausbau der Hardware ist es wichtig zu wissen, wie Hardware in der Vergangenheit ausgelastet war
* Kundenwunsch: Verfügbarkeitsreport
* Verbrauchsanalyse zur Berechnung der Ressourcen
* 5 Kategorien der Anforderungen an Monitoring-System
  + Zustand des Systems beobachten
    - „End-to-End“-Monitoring: ausgelieferte Daten so nah wie möglich am Endbenutzer auf Funktionsfähigkeit überprüfen
    - Statuserfassung der Dienste, Software und Hardware
    - Langzeitspeicherung von Informationen über die Verfügbarkeit von Diensten und Komponenten
  + Alarmierung
    - Das manuelle Eingreifen ins System verlangen
    - Einen Mitarbeiter über die Ursache eines Fehlers informieren
    - Reaktionszeiten und Fehlerbehebung dokumentieren
  + Diagnose
    - Informationen sammeln, um detaillierte Ursachenanalyse zu ermöglichen
    - Informationssammlung für Entscheidungen
  + Qualitätsmessung
    - Datensammlung über die Leistungsfähigkeit und den Durchsatz des Systems und Teilkomponenten
    - Erfassung von vereinbarten Grenzwerten und deren Einhaltung
    - Identifikation von Engpässen, Überlastungen und Implementierungsfehlern
  + Konfiguration
    - Überwachung von standardisierten Konfigurationen
    - Warnen bei Abweichungen von einem standardisierten Vorgehen
* Für ein stabiles System ist eine Konfiguration gemäß des vereinbarten Standards essenziell
* Monitoring-System sollte folgenden Systemkonfigurationen dokumentieren und bei Abweichungen alarmieren
  + Wann wurden Änderungen vorgenommen?
  + Wird die richtige (vereinbarte) Software eingesetzt? (Experimente einzelner Mitarbeiter kontrollieren)
  + Wann wurden Updates und Patches eingespielt? Monitoring sollte Version, Release einer Software dokumentieren
  + Gibt es Sicherheitsupdates für Software und das Betriebssystem und wann wurden diese Updates eingespielt?
* Einen einzelnen Webserver kann man mit ein paar Skripten überwachen
* Ein Netzwerk und Server im produktiven Einsatz zu überwachen, reichen ein paar Skripte nicht aus
* Dienst wie Cloudradar kann mehr:
  + Nicht nur das Endprodukt, sondern alle Teilkomponenten werden überwacht (Hardware, Software, Betriebssystem und Netzwerkinfrastruktur)
  + Durch Überwachen der Teilkomponenten könne Fehler vorgebeugt werden (freier Festplattenspeicher)
  + Ressourcenengpässe werden frühzeitig erkannt
  + Einheitliches Setup, Monitoring erkennt, wenn ein Kollege nicht die vereinbarte Konvention einhält
  + Alarmierung nur mit relevanten Daten, sodass der Admin sofort den Fehler beheben kann

## Quelle: wbs

* Um Ausfälle rechtzeitig zu erkennen und proaktiv zu reagieren, muss eine Infrastruktur unter permanenter Beobachtung sein
* Überwachung mit hohem personellem Aufwand stichprobenartig möglich
* Vollständige Überwachung ist daher unter hohem Automatisierungsgrad möglich
* Flexibles Monitoring:
  + Informationserfassung
    - Informationen automatisiert sammeln, systematisch erfassen und zur Auswertung ablegen
  + Analyse
    - Erfasste Daten mit Hilfe von Schwellenwerten und Sollzuständen auswerten und Ergebnisberichte konfigurieren
  + Problemerkennung
    - Fehler und Ursachen anhand gelieferter Messwerte frühzeitig erkennen und identifizieren
  + Alarmierung und Visualisierung
    - Bei Problemen benachrichtigen, Nutzer oder User Help Desk, ua per Mail oder SMS
* Vorteile von Monitoring:
  + Live Übersicht über die gesamte IT-Infrastruktur inklusive Abbildung von Abhängigkeiten
  + Automatisierte Überwachung und Benachrichtigung rund um die Uhr
    - Überwachung von Backup Prozessen
    - Netzwerk Monitoring
    - Monitoring von Servern
    - Überwachung von Kommunikationswegen
  + Trendanalyse für besseren Einsatz von Rechenleistung und Ressourcen
  + Proaktive Fehlererkennung
  + Proaktive Erkennung von Ressourcenengpässen

## Quelle: crossmedia

* Überwachung bzw laufende Kontrolle
* Verantwortliche Person muss feststellen, ob alle Komponenten innerhalb der notwendigen Parameter liegen
* Mit Monitoring-System kann Systemadministrator Fehler, Performance, Zugriffe und vieles mehr vergleichen und analysieren
* Überbegriff für alle Arten von systematischen Erfassungen durch technische Hilfsmittel oder spezielle Beobachtungssysteme
* Überwachung der Netzwerke aller Bereiche der IT, inklusive Nutzerverhalten der Anwender
  + Also auch Sicherheit und Verfügbarkeit
* Historical Monitoring
  + Proaktives Arbeiten
  + Vorrausschauendes und vorausplanendes Handeln der Admin
  + Automatische Langzeitstatistiken, um neben eigentlichem Monitoring auch Kapazitäten planen zu können
  + Dadurch mögliche Budgetplanung des Unternehmens unterstützen
  + Admin kann Missstände erkennen, Betroffene informieren und Lösungsansätze entwerfen
* Real-Time-Monitoring
  + Anwenderüberwachung von Servern bzw. Netzwerkumgebungen
  + Reaktives Arbeiten
  + Admin reagiert auf bereits eingetretene Probleme
  + Laufende Überwachung gewährleitet, dass Ausfälle erkannt und Betroffene in Kenntnis gesetzt werden können
  + Idealfall: Fehler registriert und behoben bevor der Nutzer diesen bemerkt
* Ermittlung von Ursachen, welche die Parameter außerhalb der gewünschten Bereiche bewegen
* Ausfallursachen erkennen und lokalisieren
* Sinnvolles Monitoring führt zu Kosten sparen zu können
* Durchgängiges Monitoring
  + Dauerhafte Überwachung in nahezu allen Bereichen
  + Informationen werden geliefert, die ein frühzeitiges Lokalisieren von Ausfällen ermöglichen
  + Dadurch Performance-Einbrüche proaktiv beseitigen
* Aktives Monitoring
  + Softwareroboter führen automatisiert Programme und Dienste aus, Simulation eines Endanwenders
  + Währenddessen kann Monitoring-System Fehler vorrübergehend beheben
  + Jedoch eigentliche Ursache des Problems besteht weiter und wird ggf verschleiert
  + Bei Geräteausfall kann aktives Monitoring nichts aufzeichnen
* End-to-End-Monitoring
  + Fokus auf Funktionalität eines Dienstes oder Programmes
  + Misst Nutzung am Ort der Nutzung
  + Simulieren typische Nutzerverhalten oder zeichnen Tätigkeit echter Nutzer auf
  + Ziel: Sicherung der Services aus Nutzerperspektive
  + Application Response Time Monitoring überprüft, ob Antwortzeit im angemessenen Bereich liegt
* Verfügbarkeit und Stabilität der Systemumgebung erhöhen

# Standards aus verschiedenen Sprachen

## Java

Quelle: Baeldung

* Protokollierung aktivieren
  + Bibliothek\*en hinzufügen
  + Aufbau
  + Log Anweisung platzieren
* Log4J2
  + Verbesserte Version von Log4J-Protokollierungsframework
  + Verbesserung: asynchrones Protokollieren
  + Appender bestimmt das Pattern der Lognachricht und den Ausgabeort (bspw. Konsole)
  + Appender (ein oder mehrere) wird im Root Abschnitt hinzugefügt
  + Asynchrone Protokollierung mit LMAX Disruptor -> Inter-Thread-Kommunikation
  + TODO: Nachbauen
    - <**Configuration** status="debug" name="Config" packages=""> <**Appenders**> <**Console** name="stdout" target="SYSTEM\_OUT"> <**PatternLayout** pattern="%d{yyyy-MM-dd HH:mm:ss} %p %m%n"/> </**Console**> </**Appenders**> </**Configuration**>
    - <**Appenders**> <**File** name="fout" fileName="baeldung.log" append="true"> <**PatternLayout**> <**Pattern**>%d{yyyy-MM-dd HH:mm:ss} %-5p %m%nw</**Pattern**> </**PatternLayout**> </**File**> </**Appenders**>
    - **private** **static** **Logger** logger = LogManager.getLogger(Log4jExample.class);
    - logger.debug("Debug log message"); logger.info("Info log message"); logger.error("Error log message");
    - 2016-06-16 17:02:13 INFO Info log message 2016-06-16 17:02:13 ERROR Error log message
* Logback
  + Verbesserte Version von Log4J
  + Nutzt SLF4J als Schnittstelle
* SLF4J
  + Gemeinsame Schnittstelle und Abstraktion für die meisten Java-Logging-Frameworks
  + Standardisierte API
* Log4J
  + Altehrwürdiges Framework
  + Veraltet aber Grundlage für moderne Nachfolger

Quelle: log4j2

* Anfang 1996 Entwicklung von log4j als eigene log-API von EU-Semper-Projekt
* Beliebtes Logging-Paket für Java
* Unter Apache Software License als vollwertige Open Source Lizenz
* Manchmal debugging nicht anwendbar wegen Threads sodass loggs die einzige Möglichkeit der Nachvollziehbarkeit sind
* Einmal im Code eingefügt, braucht es kein menschliches Eingreifen zur Generierung von Logs
* Logging als audit tool
* Detailliertes durchlaufen des codes weniger effizient als logs an bestimmten stellen
* Debugging ist vorrübergehend, logs können abgespeichert und später analysiert werden
* uU kann logging eine Anwendung verlangsamen
* zu ausführlich kann zu blindheit führen
* Log4j2
  + Log4j1: Schwierig wartbar, da mit alten Java Versionen kompatibel sein muss
  + Gründe für Log4J2:
    - Kann als audit logging framework genutzt werden (bei Neukonfiguration von log4j1 und logback können Ereignisse verloren gehen)
    - Asynchrone Logger für Multithread-Szenarien (Durchschlag bis zu 10 mal höher und wesentlich kleinere Latenz)
    - Garbage free für eigenständige Anwendungen, geringer Müll für Webanwendungen während stationärem Logging
    - Plugin-System -> leicht erweiterbar
    - Benutzerdefinierte Loglevel möglich (im Code und Config)
    - Unterstützt Lambda Ausdrücke zum Zusammenbauen der Log msg
    - Viele Logback Appender akzeptieren Daten nur in einem bestimmten Format, Log4j2 kann gewünschtes Format transportieren
    - Rückgabepararmeter: bei Log4j1 und Logback String und bei Log4j2 Byte-Array
    - Parallelitätsunterstützung statt Deadlock wie bei Log4j1

Quelle: oracle

* Logger Objekt durch Logger-Factory-Methode -> erstellen neuen oder gegen geeigneten Logger zurück
* Logger von getLoggerFabrikMehtode durch Garbage Collector erfassbar, falls nicht starker Verweis auf den Logger
* Logs werden durch Handler zu Ziel geleitet (Konsole, Datei, etc.)
* …..

Quelle: Rheinwerk

* Java.util.logging seit Java 1.4 aber nicht kompatibel zu log4j und nicht so leistungsstark
* Loglevel: SEVERE, WARNING; INFO; CONFIG; FINE; FINER; FINEST mit Aufruf: log(Level.SEVERE, msg)
* Log msg formatieren: bspw. Parameter Filename und File.size mit
  + log.log( Level.INFO, "Open {**0**}", **file** );
  + log.log( Level.INFO, "Open {**0**}, size {**1**}", new Object[]{ **file**, **Files.size( Path.get(file) )** } );
* Je nach Loglevel des Loggers werden Nachrichten protokolliert oder ignoriert
* JUL bietet Möglichkeit boolean isLoggable(Level level), um aufwändig aufgebaute Log-Nachrichten nicht auszuwerten falls sie nicht geloggt werden würde
  + File.size als Auswertung teuer
* Zweite Möglichkeit seit Java 8 mit Lambda: Programmcode wird erst ausgewertet, wenn Logging-Framework entschieden hat, dass der Block ausgeführt werden muss
* Neue Methoden im Logger seit Java 8: log(Level, Throwable, Supplier), severe(Supplier), warning(Supplier), etc.
* Logp() ist eine überladene Methode, die auch Klassename und Methodenname annimmt und loggt
  + Wird oft bei betreten oder verlassen von Methoden genutzt mit entering() oder existing()
  + Normale Logger-Methoden greifen auf log() zurück
* Log4j
  + Open source
  + Apache Software License
  + Populäre API in andere Sprachen übersetzt
  + Zentrale Begriffe:
    - Logger: Zentrum des Interesse und kümmert sich um Log nachrichten und ob diese geloggt werden sollen
    - Level: was soll geloggt werden, Warning oder auch Information
    - Appender: hängen Logausgabe irgendwo an, bspw Dateien
    - Layout: Format der Ausgabe, bspw HTML oder XML
  + Lässt sich mit Java Anweisungen oder config Datei konfigurieren
* SLF4J
  + Java Standardbibliothek und log4j unter einen Hut
  + Simple Logging Facade for Java
  + Einfache API vor einer komplexen Implementierung
  + Besser als Commons Logging JCL

## JavaSkript

Quelle: Stackify

* Bei Entwicklung ausgabe in konsole
* Bei veröffentlichtem code ist das schwierig
* Deshalb brauchen wir eine möglichkeit die fehlermeldungen beim client zu einem server zu senden um sie dort auszulesen zu können
* Integrierte log methoden
  + Klartext log()
  + Info
  + Warn
  + Error

## Python

Quelle: ionos

* Anwendungsbereiche von logging
  + Debugging
  + Auffinden und beheben von sicherheitslücken
  + IT-forensik: kritische Vorfälle nachvollziehen
  + IT-audit: Überprüfung auf datensicherheit und effiezienz der programme und betriebssysteme
  + Vergleich verschiedener versionen von datensätzen
* Print ausgaben an stellen wo fehler vermutet werden
  + Nachteil: nachher befehle entfernen bzw auskommentieren
  + Sonst ausgabe auch bei anwender
  + Code unaufgeräumt
* Fehler schweregrade:
  + Debug: programmdiagnose bsp unerwartete einrückung in zeile xy
  + Info: system läuft ordnungsgemäß bsp funktion x wird ausgeführt
  + Warning: weitestgehend ordnungsgemäß mit unerwarteter situation oder zukünftiges problem erwartet, bsp speicherplatz knapp
  + Error: funktion konnte nicht ausgeführt werden
  + Critical: schwerwiegendes problem
* Python logging level entsprechen statischen funktionen
* Debug ist das niedrigste level und umfasst alle anderen level
* Logging modul in pytjon bibliothek
* Logging modul enthält: logger, handler, filter und formatter
  + Logger: logging.getLogger(loggername)
  + Hanlder: nimmt infos von logger und sendet sie weiter, bspw Filehandler sendet daten an dateien
* Tool zur fehlerprävention, kontrolle nach hacker angriffen oder einfache analyse

Quelle: docs-python

* Modul enthält
  + Logger wird direkt im code verwendet
  + Handler sendet die logs ans ziel
  + Filter bietet möglichkeit ob eine Datensatz protokolliert werden soll
  + Formatter layout der logs
* Logger niemals direkt initialisieren sonder mit getLogger(\_\_name\_\_) den logger nutzen der bereits exisitiert, um auf der gleichen referenz zu arbeiten
* Genutzt über import logging
* Beim erstellen eines root logger ist das level warning
* Logger methoden
  + Debug(mg, args, kwargs)
    - Protokolliert nachricht mit level debug
    - Msg: nachrichtenformatzeichenfolge
    - Args: Argument die in msg eingefügt werden
    - Kwargs kann eines der vier Schlagwörter sein: exc\_info, stack\_info, stacklevel oder extra
      * Exc\_info: falls nicht false werden exception infos zu logs hinzugefügt
      * Stack\_info: default false, falls true wird stack an log angefügt
      * Stacklevel: default 1, falls größer werden dementsprechend viele stack frames übersprungen
      * Extra: bspw ein Wörterbuch
  + Bsp:

FORMAT = '**%(asctime)s** **%(clientip)-15s** **%(user)-8s** **%(message)s**'

logging.basicConfig(format=FORMAT)

d = {'clientip': '192.168.0.1', 'user': 'fbloggs'}

logger = logging.getLogger('tcpserver')

logger.warning('Protocol problem: **%s**', 'connection reset', extra=d)

* + Ausgabe:

2006-02-08 22:20:02,165 192.168.0.1 fbloggs Protocol problem: connection reset

* + Info(msg, args, kwargs)
  + Warning(msg, args, kwargs)
  + Error(msg, args, kwargs)
  + Critical(msg, args, kwargs)
  + Log(level, msg, args, kwargs)
* Loglevel
  + Critical 50
  + Error 40
  + Warning 30
  + Info 20
  + Debug 10
  + Notset 0
* Logging.handler
* Logging.formatter

Jaeger

* Distributed tracing system
* Open source
* Für Überwachung und Fehlerbehebung von Microservice basierten verteilten Systemen verwendet
* Verteilte Kontextweitergabe
* Verteilte transaktionsüberwachung
* Ursachenanalyse
* Dienstabhängigkeitsanalyse
* Leistungs-/Latenzoptimierung
* Kompatibel mit Go, Java, Node, Python, C++ und C#
* Mehrere Speicher Backend: Cassandra, Elasticsearch, memory
  + Cassandra und elasticsearch als open source nosql db
  + In memory Speicher zum testen von setups
* In Go implementierte Backend-Komponenten
* React/ Javascript UI -> Leistungsstark sodass große Datenmengen dargestellt werden können
* Hohe Skalierbarkeit: normalerweise mehrere Milliarden Spans pro Tag
  + Logical unit of work mit operation name, startzeit und dauer
* Jaeger ui unterstützt diagramme: system architecture und deep dependency graph
  + System achitecture:
    - zeigt abhängigkeiten zwischen zwei Anwendungen
    - Aufrufe von Services aber keine Endpunkte
  + Dependency deep graph
    - Transitive dependency graph

Signoz:

* Distributed tracing
  + Benutzeranfragen über mehrere Dienste/ Services in Microservice verfolgen
* Benötigen zentralen Überblick über die Leistung von Anfragen in allen Diensten, um zu erkennen wo ein fehler ist
* Tracing verfolgt anfrage durch jeden dienst und misst latenz
* Leistungsprobleme verbessern und endbenutzererfahrung bessern
* Signoz:
  + Full stack open source APM und Observability Tool
  + Drei säulen der Observability: Logs, Metriken und Traces
  + Einheitliche UI für metriken und traces
  + Verfolgen von
    - Benutzeranfragen pro sekunde
    - Latenzen von microservices in der anwendung
    - Fehlerrate von anfragen
    - Langsame endpunkte
    - Benutzeranfragen über mehrere services
* Jaeger
  + Open source apm(application performance management) tool
  + Verteilte Kontextweitergabe
  + Verteilte Transaktionsüberwachung
  + Ursachenanalyse
  + Dienstabhängigkeitsanalyse
  + Leistungs-/Latenzoptimierung
  + zwei beliebte Open-Source-NoSQL-Datenbanken als Trace-Speicher-Backends: Cassandra und Elasticsearch
* zipkin
  + open source apm tool
  + timing-daten zur behebung von latenzproblemen
  + jede anfrage bekommt id wodurch sie in jedem Dienst identifiziert
  + Reporter senden Daten an Zipkin
  + Kollektoren, die Ablaufverfolgungsdaten im Speicher beibehalten
  + API zum Abfragen von Daten
  + Benutzeroberfläche
  + Identifizieren von veralteten Diensten und Fehlerpfaden

Zipkin

* Behebung von latenzproblemen
* Mit trace id direkt zu Prozess springen
* Oder mit attributen wie dienst, vorgangsname etc abfragen
* Ui zeigt abhängigkeitsdiagramm
* Daten an zipkin über http
* Daten in backend cassandra oder elasticsearch gespeichert oder im arbeitsspeicher

It-aktuell

Prometheus

Grafana