Kapitel 6 Betrieb, Ueberwachung und Fehleranalyse

albion.eu www.tectrain.ch www.accso.de

< Kapitel 5 Installation und Roll Out

Kapitel 7 Case Study >

Kapitel 6 Betrieb, Ueberwachung und Fehleranalyse

FLEX Lehrplan

6 Betrieb, Überwachung und Fehleranalyse

Dauer: 90 Min Übungszeit: 30 Min

6.1 Begriffe und Konzepte

Monitoring, Operations, Logging, Tracing, Metrics.

6.2 Lernziele

6.2.1 Was sollen die Teilnehmer können?

- Die Teilnehmer sollen ein Konzept grob skizzieren und verstehen können, auf dessen Basis ein System überwacht werden kann d. h. den aktuellen Status zu beurteilen, Fehler und Abweichungen vom normalen Betrieb möglichst zu vermeiden oder zumindest so früh wie möglich zu erkennen zu behanden.
- Dabei können sie abhängig vom konkreten Projekt-Szenario den Fokus im Konzept auf Logging, Monitoring und die dazu notwendigen Daten legen.
- Die Teilnehmer sollen Architekturvorgaben so treffen können, dass der Einsatz geeigneter Werkzeuge bestmöglich unterstützt wird, dabei jedoch angemessen mit Systemressourcen umgegangen wird.

6.2.2 Was sollen die Teilnehmer verstehen?

- Logging und Monitoring kann sowohl fachliche als auch technische Daten enthalten.
- Die richtige Auswahl von Daten ist zentral für ein zuverlässiges und sinnvolles Monitoring und Logging.
- Damit Systeme, insbesondere solche, die sich aus vielen einzelnen Teilsystemen zusammensetzen, betreibbar sind, muss die Unterstützung des Betriebs mit hoher Priorität Bestandteil der Architekturkonzepte sein.
- Damit eine möglichst hohe Transparenz erreicht wird, müssen sehr viele Daten erfasst, aber auch zielgruppengerecht voraggregiert und auswertbar gemacht werden.
- Die Teilnehmer sollen verstehen, welche Informationen sie aus Log-Daten und welche sie (besser) durch Instrumentierung des Codes mit Metrik-Sonden beziehen können.
- Die Teilnehmer sollen verstehen, wie eine typische zentralistische Logdaten-Verwaltung aufgebaut ist und welche Auswirkungen sie auf die Architektur hat.
- Die Teilnehmer sollen verstehen, wie eine typische zentralistische Metriken-Pipeline aufgebaut ist (Erfassen, Sammeln & Samplen, Persistieren, Abfragen, Visualisieren) und welche Auswirkungen sie auf die Architektur hat (Performance-Overhead, Speicherverbrauch,...).
- Die Teilnehmer sollen die unterschiedlichen Möglichkeiten von Logging, Monitoring und einer Operations DB (siehe M. Nygard, Release IT!) verstehen, was man wofür einsetzt und wie man diese Werkzeuge sinnvoll kombiniert.

6.2.3 Was sollen die Teilnehmer kennen?

- Werkzeuge für zentralistische Logdaten-Verwaltung
- Werkzeuge für zentralistische Metriken-Verarbeitung
- Unterscheidung zwischen Geschäfts-, Anwendungs- und Systemmetriken
- Bedeutung wichtiger, werkzeugunabhängiger System- und Anwendungsmetriken

6.3 Referenzen

- Eberhard Wolff: Continuous Delivery: Continuous Delivery: Der pragmatische Einstieg, dpunkt, 2014, ISBN 978-3-86490-208-6
- Michael Nygard: Release It!: Design and Deploy Production-Ready Software, Pragmatic Programmers, 2007, ISBN 978-0-97873-921-8

Inhalte

(A) Anforderungen - "Design for Operations"

1. Design for Operations

Im Design von Anwendungen muss der **fachliche wie technische Betrieb** von Tag 1 an mitgedacht werden. Dabei sind insbesondere die folgenden Aspekte zu berücksichtigen:

- **Skalierbarkeit:** Die Fähigkeit des Systems, eine wachsende Zahl von Benutzern oder Anfragen ohne Leistungseinbußen zu bewältigen.
- **Verfügbarkeit:** Der Prozentsatz der Zeit, in der das System betriebsbereit und für die Benutzer zugänglich ist.
- **Ausfallsicherheit:** Die Fähigkeit des Systems, sich von Ausfällen zu erholen und den Betrieb aufrechtzuerhalten.
- **Leistung:** Die Geschwindigkeit und Reaktionsfähigkeit des Systems auf Benutzeranfragen.
- **Sicherheit:** Die Fähigkeit des Systems, vor unbefugtem Zugriff, Verstößen und Datenverlusten zu schützen.
- **Nachvollziehbarkeit:** Die Fähigkeit, die Leistung, Nutzung und Fehlerprotokolle des Systems zu überwachen, um Probleme zu erkennen und zu beheben.
- **Testbarkeit** und **Automatisierung:** Die Fähigkeit, Aufgaben und Prozesse und Tests zu automatisieren, um menschliche Eingriffe zu reduzieren und die Effizienz zu verbessern.

Es ist auch wichtig zu überlegen, wie das System langfristig eingesetzt, verwaltet und gewartet werden soll. Dazu gehören Faktoren wie einfache Bereitstellung, Aufrüstbarkeit und die Verfügbarkeit von Dokumentation und Support.

Flexibilität durch Modularisierung und vor allem Verteilung in Services hat seinen Preis, vor allem in Form erhöhter Aufwände im Betrieb. Ein verteiltes System ist flexibel und skalierbar, lässt sich aber nur mit erhöhten Aufwänden betreiben, weswegen Automatisierung, Fehlerkorrekturen, Nachvollziehbarkeit etc. hier direkt mit gedacht werden müssen. Besonders relevante Qualitätseigenschaften sind dabei Skalierbarkeit, Verfügbarkeit, Zuverlässigkeit, Fragen von Konsistenz und Sicherheit, Performance und Durchsatz. Hier gilt es, im Anforderungsprozess auch direkt von Seiten des Betriebs zu erheben, abzustimmen, zu implementieren und zu testen.

2. Was sind die Hauptunterschiede zwischen Monolithen, Modulithen und verteilten Systemen in Bezug auf betriebliche Aspekte?

Die Hauptunterschiede zwischen Monolithen, Modulithen und verteilten Systemen in Bezug auf den **Betrieb** sind folgende:

- **Monolith:** In einer monolithischen Architektur sind alle Komponenten des Systems eng integriert und werden zusammen als eine Einheit eingesetzt, damit auch als Einheit betrieben.
- **Modulith:** In einer Modulith-Architektur besteht das System aus mehreren lose gekoppelten Modulen, aber alle Module werden zusammen als eine Einheit eingesetzt. Ein Modulith verhält sich damit im Betrieb wie ein Monolith.
- **Verteiltes System:** In einer verteilten Architektur besteht das System aus mehreren unabhängigen Knoten, die über ein Netz kommunizieren.

Die typischen Betriebsaufgaben unterscheiden sich in der Komplexität und im Umfang:

- Deployment, Skalierung, Überwachung, Erfassung von Metriken, Updates, Support ist in einem monolithischen System leichter.
- Dagegen skaliert es in der Regel nicht (so gut, siehe auch Scale Up vs Out unten).
- Auch ist die Verfügbarkeit eines Monolithen begrenzt.

3. Auswirkungen eines "betriebsorientierten" Entwurfs auf die Anwendung: Wie kann man den Code sauber halten?

Das Design von **betrieblichen Aspekten hat einen Einfluss auf die Codestrukturen und -qualität** einer Anwendung. Ein gut konzipiertes System, das die oben aufgeführten Qualitätsmerkmale berücksichtigt, ist robuster, zuverlässiger und wartungsfreundlicher.

Typische Methoden, um den Code dennoch "sauber" zu halten, sind:

- Verwendung eines modularen Designs, bei dem verschiedene Funktionen in unterschiedliche Module oder Komponenten aufgeteilt werden, die unabhängig voneinander leicht verstanden, getestet und gewartet werden können.
- Dies kann auch dazu beitragen, die Skalierbarkeit und Wartbarkeit zu verbessern, da es einfacher ist, Funktionen hinzuzufügen oder zu ersetzen, ohne den Rest des Systems zu beeinträchtigen.
- Außerdem ist es hilfreich, klare und konsistente Namenskonventionen zu verwenden, einen einheitlichen Codestil zu befolgen und automatisierte Tools zur Überprüfung der Codequalität und von Fehlern einzusetzen.

Dabei sollte man den fachlichen Business Code von technischen Details und betrieblichen Funktionen trennen, wie z. B. über:

- Die **aspektorientierte Programmierung (AOP)** ist ein Programmierparadigma, das die Trennung von übergreifenden Belangen wie Protokollierung, Sicherheit und Fehlerbehandlung von der Kerngeschäftslogik einer Anwendung ermöglicht. AOP kann dazu beitragen, den Code sauber zu halten, indem es die Identifizierung und Verwaltung dieser querschnittlichen Belange ermöglicht und die Menge an dupliziertem Code reduziert.
- Das Konzept von **Interceptoren** kann ebenfalls helfen, den Code sauber zu halten, in dem an zentraler Stelle übergreifende Funktionen wie Protokollierung und Sicherheit zentralisiert und wiederverwendbar implementiert sind und nicht den fachlichen Code "verschmutzen".

AOP und Interceptoren sind mit Bedacht einzusetzen. Die "Magie" hinter solchen Strukturen passiert erst zur Laufzeit, z.B. per Reflection oder Dynamic Proxies. Damit lassen sich die nötigen Abhängigkeiten nicht (immer) erkennen, wenn z.B. statische Code-Analysen oder -Reviews erfolgen. Auch die Analyse von Fehlern (z.B. über Stacktraces) ist aufwendiger.

(B) Betrieb der Anwendung

1. Betriebsaufgaben

In Bezug auf DevOps und cross-funktionale Teams ist es üblich, dass die Verantwortung für diese Aufgaben unter den Teammitgliedern aufgeteilt wird (siehe Kapitel 5 (B)).

So können beispielsweise Entwickler für das Schreiben von Code verantwortlich sein, der sich leicht überwachen und automatisieren lässt, während sich Betriebsingenieure auf die Implementierung von Überwachungs- und Automatisierungstools konzentrieren können. Darüber hinaus ist es üblich, dass ein Team einen zentralen Ansprechpartner für alle betriebsbezogenen Aufgaben hat, der dafür verantwortlich ist, dass alle Aufgaben ausgeführt werden und dass eine klare Kommunikation und Koordination im Team stattfindet. Wichtig ist auch ein klarer Prozess für das Management von Betriebs-Incidents, und das Team sollte klare Rollen und Verantwortlichkeiten für Notfälle festlegen.

Der Betrieb einer Anwendung umfasst in der Regel eine Vielzahl von **Betriebsaufgaben,** um sicherzustellen, dass die Anwendung reibungslos läuft und die Anforderungen der Benutzer erfüllt, darunter:

- **Maintenance/Wartung:** Regelmäßiges Einspielen von Updates und Sicherheitspatches für die Anwendung und ihre Abhängigkeiten.
- Das **Deployment** umfasst in der Regel das Deployment der gesamten Anwendung auf einen neuen Server oder das Update der Anwendung auf einem vorhandenen Server.
- **Überwachung,** siehe unten
- **Backup** und **Recovery:** Regelmäßige Sicherung der Anwendungsdaten und Konfiguration von Disaster-Recovery-Verfahren.
- **Skalierung:** Verwaltung der Kapazität der Anwendung zur Bewältigung einer steigenden Last, z. B. durch Hinzufügen weiterer Server oder Anpassung der Anwendungskonfiguration.
- **Automatisierung:** Automatisierung sich wiederholender Aufgaben, wie z. B. Bereitstellungen und Backups, ist sinnvoll, um die Effizienz zu steigern und das Risiko menschlicher Fehler zu verringern.
- **Fehlersuche:** Untersuchung und Behebung von Problemen, die in der Anwendung auftreten, z. B. Bugs und Leistungsengpässe.
- **Unterstützung:** 1st, 2nd, 3rd Level Support - siehe unten

2. Monitoring, Logging, Tracing

Für die Überwachung des Systems kommen v.a. zum Einsatz:

Monitoring:

- Kontinuierliche Überwachung der Leistung der Anwendung, einschließlich CPU- und Speichernutzung, Antwortzeiten und Fehlerraten.
- Dies schliesst technische Kennzahlen wie auch fachliche Kennzahlen (KPIs) mit ein.
- In der Regel wird Monitoring durchgeführt als Mischung aus
  - Kurzfrist- (aktuelle Ereignisse, aktuelles Systemverhalten) und
  - Langfristdarstellung (Verhalten und Änderungen über längeren Zeitraum wie Stunden oder Tage hinweg).

Logging:

- In einer Anwendung umfasst das Logging in der Regel das Sammeln und Speichern von Informationen über das Verhalten und die Leistung der Anwendung.
- Dies kann Informationen wie Anwendungsprotokolle, Systemprotokolle und Zugriffsprotokolle umfassen.
- Logs werden zur Fehlersuche, Fehlerbehebung und Prüfung verwendet.
- Diese Logs werden in der Regel an einem zentralen Ort gespeichert, z. B. auf einem Protokollserver oder in einer Datenbank.

Tracing:

- In einer Anwendung umfasst das Tracing in der Regel die Verfolgung des Flusses einer Anfrage durch die Anwendung und die Identifizierung von Engpässen oder Fehlern.
- Dies kann Informationen wie Anforderungs- und Antwortzeiten und Call Stacks umfassen.
- Tracing kann zum Debugging, zur Fehlerbehebung und zur Leistungsoptimierung verwendet werden.
- Die Traces werden in der Regel an einem zentralen Ort gespeichert, z. B. auf einem Trace-Server oder in einer Datenbank

3. Typische Aufgaben im Betrieb für 1st, 2nd und 3rd Level Support

Der **1st-Level-Support** ist die erste Anlaufstelle für Benutzer, die Probleme mit der Anwendung oder dem System haben. Seine Hauptaufgaben:

- Beantwortung von Benutzeranfragen und Behebung von grundlegenden Problemen
- Sammeln von Informationen über das Problem und ggf. Weiterleitung an das zuständige Team
- Bereitstellung von grundlegenden Anweisungen und Anleitungen für die Nutzung der Anwendung oder des Systems

Der **2nd-Level-Support** ist für die Bearbeitung komplexerer Probleme zuständig, die vom 1st-Level-Support nicht gelöst werden. Seine Hauptaufgaben:

- Fehlersuche und Behebung fortgeschrittener technischer Probleme
- Bereitstellung von technischem Fachwissen und Anleitung für den 1st-Level-Support
- Untersuchen und Diagnostizieren von Problemen
- Koordinierung mit anderen Teams oder Anbietern zur Lösung von Problemen

Der **3rd-Level-Support** ist für die Bearbeitung der komplexesten Probleme und deren Behebung an der Wurzel zuständig. Seine Hauptaufgaben:

- Bereitstellung von fundiertem technischem Fachwissen und Anleitung für den 2nd-Level-Support
- Untersuchung und Behebung von Problemen an der Wurzel
- Entwicklung und Umsetzung von Lösungen, um zu verhindern, dass ähnliche Probleme in Zukunft auftreten
- Mitwirkung an der Konzeption und Implementierung neuer Systeme und Anwendungen

Die Support-Level "verschwimmen" im typischen DevOps-Organiationsmodell, insbesondere zwischen 2nd- und 3rd-Level-Support, da dort cross-funktional zusammengearbeitet wird - sowohl zwischen Betrieb und Entwicklung als auch zwischen technischem und fachlichem Betrieb.

4. Zuständigkeiten des "Fachlichen" und "Technischen" Betriebs

Der Betrieb ist dafür verantwortlich, dass das Tagesgeschäft eines Unternehmens reibungslos und effizient abläuft. Dazu gehören die Verwaltung und Wartung der Systeme und Prozesse, die das Unternehmen unterstützen, sowie die Überwachung der Mitarbeiter und Ressourcen, die für die Durchführung dieser Aktivitäten benötigt werden.

- Dabei bezieht sich der **"Fachliche Betrieb"** auf den funktionalen oder technischen Betrieb des Unternehmens, wie z. B. das Management von Geschäftsprozessen, Kundendienst und die Koordination von Ressourcen.
- Hingegen umfasst der **"Technische Betrieb"** die technischen oder IT-nahen Betriebsaufgaben des Unternehmens, z. B. die Verwaltung von IT-Systemen und -Infrastruktur, einschließlich Hardware, Software und Netzwerken.

Solche Aufgaben können in einem klassischen Betriebsteam umgesetzt und ausgeführt werden (z.B. bei großen, formal operierenden Unternehmen und Behörden). Alternativ ist sind Platform Engineering und Dev Teams im Sinne von DevOps für den Betrieb der Anwendung verantwortlich ("You build it, you run it").

An den inhaltlichen Aufgaben ändert sich nichts durch ein anderes organisatorisches Modell - die Aufgaben werden nur anders, in modernen Organisationen kollaborativ und gemeinsam ("cross-funktional") gelöst.

(C) Service Level Agreements, Incident Metrics

Info
https://www.atlassian.com/incident-management/kpis/common-metrics

https://www.informatik-aktuell.de/betrieb/verfuegbarkeit/hochverfuegbarkeit-und-downtime-eine-einfuehrung.html

https://www.informatik-aktuell.de/betrieb/verfuegbarkeit/hochverfuegbarkeit-und-downtime-metriken.html

1. SLAs

Service Level Agreements (SLAs) sind Vereinbarungen zwischen dem Anbieter eines Dienstes (z. B. einer Anwendung) und dem Kunden, in denen festgelegt wird, welche Leistungen der Kunde erwarten kann. Zu den typischen SLAs einer Anwendung im Betrieb gehören:

- **Verfügbarkeit:** Der prozentuale Anteil der Zeit, in der die Anwendung zugänglich und betriebsbereit ist, oft auf monatlicher oder jährlicher Basis gemessen.
- **Reaktionszeit:** Die Zeit, die die Anwendung benötigt, um auf eine Benutzeranfrage zu reagieren, oft gemessen in Sekunden oder Millisekunden.
- **Durchsatz:** Die Anzahl der Anfragen, die die Anwendung pro Sekunde oder pro Minute bearbeiten kann.
- **Wiederherstellungszeit:** Die Zeit, die die Anwendung benötigt, um sich von einem Fehler oder Ausfall zu erholen, oft in Minuten oder Stunden gemessen.
- **Sicherheit:** Das Maß an Sicherheit, das die Anwendung bietet, z. B. die Einhaltung von Industriestandards oder Zertifizierungen.
- **Support:** Das Maß an Unterstützung, das den Kunden geboten wird, z. B. die Verfügbarkeit eines Helpdesks oder die Reaktionszeit auf Support-Anfragen.

SLAs sind für den Betrieb wichtig, weil sie eine klare und messbare Möglichkeit bieten, die Leistung einer Anwendung zu bewerten und Bereiche zu identifizieren, in denen Verbesserungen nötig sind. Außerdem bieten sie den Kunden die Möglichkeit, den Anbieter für die Qualität der erbrachten Leistungen zur Rechenschaft zu ziehen.

2. Best Practices für das Monitoring von Performance und Verfügbarkeit

Diese **Best Practices** empfehlen sich für das Monitoring von Performance und Verfügbarkeit:

- Überwachung wichtiger technischer **Leistungsindikatoren** wie Antwortzeiten, Fehlerraten und Ressourcenauslastung. Dies hilft, Probleme der Anwendung schnell und frühzeitig zu erkennen und zu beheben.
- Verwendung eines zentralen **Logging-Systems,** um Protokolldaten aus allen Teilen der Anwendung zu sammeln und zu aggregieren. Dies erleichtert das Durchsuchen und Analysieren von Protokolldaten sowie das Erkennen von Mustern und Trends im Verhalten der Anwendung.
- (Automatisierte) **Alerts** zur Benachrictigung, wenn bestimmte Schwellenwerte überschritten werden oder bestimmte Ereignisse eintreten. So werden sich anbahnende Probleme schneller erkannt und Ausfallzeiten minimiert.
- Hilfreich kann auch eine **Kombination aus synthetischer und realer Benutzerüberwachung** sein, um ein vollständiges Bild von der Leistung und Verfügbarkeit der Anwendung zu erhalten. Die synthetische Überwachung simuliert reale Benutzerinteraktionen mit der Anwendung, während die reale Überwachung Daten aus tatsächlichen Benutzerinteraktionen verwendet.
- Einsatz eines **Monitoring-Tools,** das verschiedene Datentypen wie Protokolle, Traces und Metriken korreliert und die Daten in Echtzeit analysieren kann. Es sollte flexibel und erweiterbar sein und sich leicht mit anderen Systemen und Tools integrieren lassen.
- Idealerweise lassen sich die Monitoring-Informationen auf einem gemeinsamen **Dashboard** darstellen und aggregieren.

Weiterhin:

- Ein Sicherheitsplan sollte vorhanden sein, der die Überwachung verdächtiger Aktivitäten, insbesondere Zugriffe, und von Sicherheitsverletzungen umfasst.
- Ein Plan für die Skalierung sollte vorhanden sein, einschließlich der Überwachung der Ressourcen und der Leistung der Server sowie des Zustands des Netzwerks und der zugrunde liegenden Infrastruktur. Idealerweise ist das bei dynamischer Skalierung automatisiert gelöst.
- Ein Notfallplan sollte vorhanden sein, der alle Überwachungssysteme umfasst, die Ausfälle erkennen und beheben können.

3. Uptime und Verfügbarkeit

Uptime und Verfügbarkeit sind zwei verwandte, aber unterschiedliche **Messgrößen für die Leistung einer Anwendung**.

- Die **Uptime** bezieht sich auf die Zeit, in der eine Anwendung betriebsbereit ist, d.h. zugänglich und einsatzfähig. Sie wird in der Regel als Prozentsatz der Gesamtzeit ausgedrückt und ist ein Maß für die Zuverlässigkeit einer Anwendung. Wenn eine Anwendung beispielsweise eine Uptime von 99,9 % hat, bedeutet dies, dass sie nur 0,1 % der Zeit nicht verfügbar ist.
- Die **Verfügbarkeit** hingegen ist ein Maß für die Nutzbarkeit einer Anwendung. Dabei wird nicht nur die Zeit berücksichtigt, in der eine Anwendung verfügbar ist, sondern auch die Zeit, die die Anwendung benötigt, um auf Benutzeranfragen zu reagieren. Sie wird in der Regel auch als Prozentsatz der Gesamtzeit ausgedrückt. Wenn eine Anwendung beispielsweise eine Verfügbarkeit von 99,9 % hat, bedeutet dies, dass sie nur 0,1 % der Zeit nicht verfügbar ist oder nicht reagiert.

Der Unterschied zwischen Uptime und Verfügbarkeit besteht also darin, dass eine Anwendung zwar verfügbar sein kann, aber aufgrund von Leistungsproblemen nicht genutzt werden kann, z. B. wenn die Antwortzeit hoch ist oder der Dienst langsam ist, so dass er für den Benutzer nicht nutzbar ist.

Sowohl die Uptime als auch die Verfügbarkeit sind wichtige Maßstäbe für die Leistung einer Anwendung im Betrieb. Hohe Uptime und Verfügbarkeit sind entscheidend, um sicherzustellen, dass eine Anwendung zuverlässig und für die Benutzer zugänglich ist.

3. Verfügbarkeit, Verfügbarkeitsklassen

Die **Availability** / **Verfügbarkeit** ist ein Maß dafür, wie oft ein System oder eine Komponente in der Lage ist, seine beabsichtigte Funktion zu erfüllen.

Verfügbarkeit ist ein Verhältnis zwischen der Zeit, in der das System oder die Komponente betriebsbereit ist, und der Gesamtzeit. Sie wird als Prozentwert ausgedrückt, wobei ein Wert von 100 % bedeutet, dass das System oder die Komponente immer betriebsbereit ist.

Availability := MTTF / (MTTF + MTTR)

Es ist wichtig zu beachten, dass diese Formel davon ausgeht, dass die Ausfallrate im Laufe der Zeit konstant ist und das System oder die Komponente nur einmal ausfallen kann, und dass das System reparierbar ist. In Wirklichkeit können Systeme mehrfach ausfallen und ihre Ausfallrate kann sich im Laufe der Zeit ändern. Außerdem berücksichtigt die Formel keine anderen Faktoren wie die Komplexität der Reparatur oder die Verfügbarkeit von Ersatzteilen. Daher ist es wichtig, diese Formel als grobe Schätzung zu verwenden und andere Faktoren zu berücksichtigen, die die Verfügbarkeit des Systems oder der Komponente beeinflussen könnten.

Info
https://www.recast-it.com/themen/verfuegbarkeitsklassen/

Über die Verfügbarkeitszahl lässt sich ein System in eine **Verfügbarkeitsklasse** (VK) einteilen, in das sog. **9er-System:** Je mehr Neuer in der Prozentangabe (99%, 99,9%, ... 99, 9999%, ...), desto weniger Ausfallzeit pro Jahr. Es gibt alternative Kategorisierungen des BSI bzw. als Availability Environment Classification.

4. Mean Time To Failure

Mean Time To Failure (MTTF) ist ein **Maß für die Zuverlässigkeit eines Systems** oder einer Komponente. Sie beschreibt die **durchschnittliche Zeit, in der ein System oder eine Komponente erwartungsgemäß ohne Ausfall** funktioniert. MTTF wird normalerweise in Stunden, Tagen oder Jahren gemessen.

Der MTTF-Wert wird berechnet, indem die **Zeit zwischen den Ausfällen eines Systems** oder einer Komponente gemessen und dann der Durchschnitt dieser Werte gebildet wird. Dieser Wert wird als Vorhersage für die **erwartete Lebensdauer** des Systems oder der Komponente verwendet. Er gibt eine Vorstellung davon, wie lange das System oder die Komponente voraussichtlich halten wird, wobei davon ausgegangen wird, dass die Ausfallrate im Laufe der Zeit konstant ist.

MTTF ist dabei ein statistisches Maß, das die Art des Ausfalls nicht berücksichtigt und keine Informationen über den Zeitrahmen liefert, in dem ein Ausfall wahrscheinlich eintreten wird. Sie berücksichtigt auch nicht die Möglichkeit von Mehrfachausfällen. Daher sollte sie in Kombination mit anderen Messgrößen verwendet werden, um ein vollständigeres Bild der Zuverlässigkeit des Systems oder der Komponente zu erhalten.

5. Mean Time To Repair

Mean Time To Repair (MTTR) ist ein **Maß für die Wartungsfähigkeit eines Systems** oder einer Komponente. Es handelt sich um die **durchschnittliche Zeit**, die **für die Reparatur eines Systems** oder einer Komponente benötigt wird, nachdem ein Fehler aufgetreten ist. MTTR wird normalerweise in Stunden, Tagen oder Jahren gemessen.

Der MTTR-Wert wird berechnet, indem die **Zeit gemessen wird, die für die Reparatur eines Systems oder einer Komponente nach dem Auftreten eines Fehlers benötigt wird**, und dann der Durchschnitt dieser Werte genommen wird. Dieser Wert wird als Vorhersage für die zu erwartende Ausfallzeit des Systems oder der Komponente verwendet, wobei davon ausgegangen wird, dass die Ausfallrate im Laufe der Zeit konstant ist. Er gibt eine Vorstellung davon, wie lange das System oder die Komponente nach einem Ausfall voraussichtlich außer Betrieb sein wird, und er ist ein Indikator dafür, wie schnell das System oder die Komponente wieder in den Normalbetrieb überführt werden kann.

Auch die MTTR ist ein statistisches Maß, das die Komplexität der Reparatur oder die Verfügbarkeit von Ersatzteilen nicht berücksichtigt und keine Informationen über den Zeitrahmen liefert, in dem eine Reparatur wahrscheinlich durchgeführt werden kann. Sie berücksichtigt auch nicht die Möglichkeit von Mehrfachausfällen. Daher sollte sie in Kombination mit anderen Metriken verwendet werden, um ein vollständigeres Bild zu erhalten.

(D) Restart

1. Geplanter Restart

Der **Restart** einer Anwendung ist von entscheidender Bedeutung, da erst in vielen Fällen erst dadurch Aktualisierungen, Fehlerbehebungen und andere Änderungen wirksam werden können.

Um sicherzustellen, dass die **Restart-Zeit** so kurz wie möglich ist, gibt es mehrere Best Practices und Strategien, die angewendet werden können:

- Die Architektur sollte **zustandslos** sein: Zustandslose Anwendungen können leicht neu gestartet werden, ohne dass Daten verloren gehen. Dies macht es einfacher, die Anwendung nach einem Neustart schnell wieder online zu bringen.
- Implementierung einer **rollenden Aktualisierungsstrategie:** Mit einer rollenden Aktualisierungsstrategie kann eine Anwendung auf einem Server nach dem anderen aktualisiert werden (siehe Kapitel 5 (F), 8.).
- **Containerisierung** ermöglicht eine schnellere und effizientere Bereitstellung von Anwendungen. Sie ermöglicht auch eine einfache Skalierung von Anwendungen und erleichtert es, die Anwendung nach einem Neustart schnell wieder online zu bringen.
- **Load Balancer** leiten den Datenverkehr an die verfügbaren Instanzen der Anwendung weiter, was einen schnelleren Neustart ermöglicht, indem die Last auf mehrere Instanzen verteilt wird.
- Der Neustartprozess muss **getestet** und **geübt** werden, um mögliche Probleme nicht erst im echten Betrieb zu erkennen.
- Die **Überwachung der Systemleistung** während des Neustarts hilft, Engpässe zu erkennen, die zu Verzögerungen beim Neustart führen können.

2. Ungeplanter Restart nach Fehler oder Crash der Anwendung

Der Neustart einer Anwendung nach einem Fehler oder Absturz ist entscheidend, damit nach Behebung des Fehlers die Anwendung weiterlaufen kann.

Das sind einige Strategien, die einen schnellen und effizienten Neustart nach einem Fehler oder Absturz gewährleisten:

- Einen **automatischer Neustart** einrichten, um die Anwendung nach einem Absturz oder Fehler automatisch wieder online zu bringen. Dies kann mit Tools wie systemd unter Linux oder launchd unter macOS erfolgen.
- **Monitoring** der Anwendung, Logging, Tracing: Eine ausreichende Protokollierung und Überwachung der Anwendung kann dazu beitragen, die Ursache des Absturzes oder Fehlers schneller zu ermitteln und entsprechende Maßnahmen zu ergreifen.
- Verwendung von Mechanismen zur **Fehlerbehandlung:** Die Implementierung von Mechanismen zur Fehlerbehandlung im Anwendungscode kann helfen, Abstürze zu verhindern und Fehler schneller zu beheben. Vgl. Resilience-Patterns in Kapitel 8.
- Einen **Notfallwiederherstellungsplan** erstellen - als Reihe von Verfahren und Richtlinien für die Wiederherstellung einer Anwendung nach einem Absturz oder Fehler. Ein solcher Plan kann dazu beitragen, Ausfallzeiten zu minimieren und eine schnelle Wiederherstellung zu gewährleisten.
- **Selbstheilungsmechanismen** können Fehler oder Abstürze automatisch erkennen und beheben, ohne dass ein manuelles Eingreifen erforderlich ist.

(E) Monitoring

1. Was ist Monitoring?

Monitoring im Betrieb bezieht sich auf den Prozess der **Erfassung und Analyse von Daten über die Leistung, Nutzung und den Zustand einer Anwendung** oder eines Systems. Diese Daten können verwendet werden, um Probleme zu erkennen, die Leistung zu messen und die Kapazität und Skalierung zu planen.

Es gibt verschiedene **Arten des Monitorings**, die je nach Anwendung und den Anforderungen des Unternehmens eingesetzt werden können:

- **Leistungsüberwachung:** Diese Art des Monitorings konzentriert sich auf die Leistung einer Anwendung oder eines Systems, einschließlich Metriken wie Antwortzeit, Durchsatz und Ressourcennutzung (z. B. CPU, Speicher, Netzwerk).
- Überwachung der **Verfügbarkeit:** Diese Art des Monitorings konzentriert sich auf die Verfügbarkeit einer Anwendung oder eines Systems, einschließlich Metriken wie Betriebszeit, Ausfallzeit und Reaktionszeit.
- **Log-Überwachung:** Diese Art des Monitorings konzentriert sich auf die Analyse der von einer Anwendung oder einem System erzeugten Protokolldateien, einschließlich Fehlern, Warnungen und anderen Ereignissen.
- **Ereignis-Überwachung:** Diese Art des Monitorings konzentriert sich auf die Verfolgung bestimmter Ereignisse oder Bedingungen, die innerhalb einer Anwendung oder eines Systems auftreten.

Monitoring kann mit verschiedenen Tools durchgeführt werden, z. B. mit Protokollanalysatoren, Software für die Leistungsüberwachung und speziellen Überwachungsplattformen.

- Die gesammelten Daten können in Dashboards visualisiert werden.
- Es können Alerts eingerichtet werden, um zu benachrichtigen, wenn bestimmte Schwellenwerte erreicht werden
- Sie können zur Fehlerbehebung, zur Erkennung von Trends und Kapazitätsproblemen verwendet werden
- Sie können zur Planung künftiger Upgrades und Skalierungen verwendet werden.

Dabei ist es wichtig, zwischen **Echtzeitüberwachung** und **historischer Überwachung** zu unterscheiden: Die Echtzeit-Überwachung dient dazu, Probleme sofort zu erkennen und darauf zu reagieren, während die historische Überwachung es ermöglicht, die vergangene Performance zu analysieren und dadurch Trends zu erkennen.

2. Wie lassen sich Fehler und unerwünschte Situationen beim Monitoring feststellen? Konkrete Fehlerszenarien vs. lang anhaltende Trends

Es gibt verschiedene Möglichkeiten, **Fehler und unerwünschte Situationen durch Monitoring** zu ermitteln:

1. **Fehlerprotokolle**, die von einer Anwendung oder einem System erzeugt werden, können analysiert werden, um bestimmte Fehler und Ausnahmen zu identifizieren, die aufgetreten sind. Diese Protokolle können detaillierte Informationen über die Ursache eines Fehlers liefern und zur Fehlersuche und -behebung verwendet werden.
2. **Leistungsmetriken** wie Antwortzeit, Durchsatz und Ressourcennutzung können überwacht werden, um Probleme zu identifizieren, die die Leistung einer Anwendung oder eines Systems beeinträchtigen können. Wenn beispielsweise die Antwortzeit einer Anwendung deutlich ansteigt, kann dies ein Hinweis auf ein Problem mit der Anwendung oder der zugrunde liegenden Infrastruktur sein.
3. **Warnungen** können eingerichtet werden, um zu benachrichtigen, wenn bestimmte Bedingungen oder Schwellenwerte erreicht werden. Wenn beispielsweise die CPU-Auslastung eines Servers einen bestimmten Schwellenwert überschreitet, kann ein Alert an das Betriebsteam gesendet werden, damit dieses Maßnahmen ergreift.
4. Mit Hilfe **statistischer Analyse** lassen sich Muster oder Trends in den Daten erkennen, die auf ein zugrunde liegendes Problem hindeuten können. Wenn zum Beispiel die Anzahl der Fehler in einer Anwendung im Laufe der Zeit zunimmt, weist dies auf ein grundliegendes Problem hin.

Dabei ist zu unterscheiden:

- **Konkrete Fehlerszenarien** beziehen sich auf spezifische, isolierte Ereignisse, die innerhalb einer Anwendung oder eines Systems auftreten. Diese Art von Fehlern sind in der Regel leicht zu identifizieren und zu beheben, da sie oft eine klare Ursache und Wirkung haben.
- **Lang anhaltende Trends** beziehen sich auf Muster oder Trends, die über einen längeren Zeitraum hinweg auftreten. Diese Art von Fehlern ist unter Umständen schwieriger zu erkennen, da sie nicht sofort offensichtlich sind, aber sie können erhebliche Auswirkungen auf die Leistung und Verfügbarkeit einer Anwendung haben. Es ist wichtig, die Daten im Zeitverlauf zu analysieren, um diese Trends zu erkennen und geeignete Maßnahmen zu ergreifen, um sie zu beheben.

Mit einer Kombination aus diesen Techniken lassen sich Fehler und unerwünschte Situationen beim Monitoring am besten erkennen.

3. Überblick über nützliche Überwachungs- und Betriebstools, insbesondere im Hinblick auf Java- und Spring-Boot-Anwendungen

Diese **Überwachungs- und Betriebswerkzeuge** werden für Java- und Spring Boot-Anwendungen typischerweise verwendet:

- Spring Boot **Actuator:** Hierbei handelt es sich um ein integriertes Tool, das mit Spring Boot geliefert wird und mehrere Endpunkte für Monitoring und Verwaltung einer Spring Boot-Anwendung bereitstellt, z. B. Zustandsprüfung, Metriken und Umgebungsinformationen.

- Java Management Extensions (**JMX**): JMX ist ein Java-Standard für die Verwaltung und Überwachung von Java-Anwendungen und -Ressourcen. JMX kann zur Überwachung von Metriken wie Speichernutzung, Thread-Anzahl und CPU-Auslastung verwendet werden.
- **Prometheus:** Ein Überwachungssystem und eine Zeitreihendatenbank, die Metriken von einer Vielzahl von Anwendungen, einschließlich Java- und Spring Boot-Anwendungen, abrufen und für eine spätere Analyse speichern kann.
- **Grafana:** Grafana ist ein beliebtes Open-Source-Tool zur Visualisierung von Daten und kann verwendet werden, um Metriken aus Prometheus und anderen Überwachungssystemen in einem benutzerfreundlichen Format anzuzeigen.
- Für das Logging bietet Spring Boot Unterstützung für verschiedene **Logging-Frameworks,** wie z. B. Logback und Log4j.

4. Prometheus, Grafana und Micrometer für Java/Spring-Boot-Anwendungen kombinieren und einrichten

Durch die Kombination von Micrometer, Prometheus und Grafana kann man eine Spring Boot-Anwendung in Echtzeit überwachen und Fehler beheben und Maßnahmen ergreifen, wenn Probleme auftreten.

Prometheus, Grafana und Micrometer können zusammen verwendet werden, um Spring Boot-Anwendungen zu überwachen:

- **Micrometer** ist eine Metrikbibliothek für Java, die über eine integrierte Unterstützung für den Export von Metriken nach Prometheus verfügt. So können Metriken aus einer Spring Boot-Anwendung gesammelt und zur Speicherung und Analyse nach Prometheus exportiert werden.
- **Prometheus** ist ein Überwachungssystem und eine Zeitseriendatenbank, die Metriken aus einer Vielzahl von Quellen, einschließlich Micrometer, abrufen kann. Prometheus kann so konfiguriert werden, dass es Metriken aus einer Spring Boot-Anwendung abruft und sie für eine spätere Analyse speichert.
- **Grafana** ist ein beliebtes Open-Source-Tool zur Visualisierung von Daten. Es kann verwendet werden, um Metriken aus Prometheus in einem benutzerfreundlichen Format anzuzeigen, wodurch Trends, Ausreißer und andere Probleme leicht zu erkennen sind.

Damit eine **Spring Boot-Anwendung** Tools wie **Prometheus und Grafana** verwendet, ist folgendes tun:

- Anwendung
  - Die passende Micrometer-Abhängigkeit zur pom.xml- oder build.gradle-Datei des Projekts hinzufügen. Um Prometheus-Unterstützung einzubinden, wird micrometer-registry-prometheus benötigt.
  - In der application.properties/.yml-Datei Konfigurationen für den Export von Metriken an Prometheus hinzufügen.
  - Mit Spring Boot Actuator den Endpunkt für die Erfassung von Metriken aktivieren. Spring Boot Actuator bietet eine Reihe von Endpunkten zur Überwachung und Verwaltung einer Anwendung. Über den Endpunkt /actuator/prometheus können Metriken nach Prometheus exportiert werden.
- Prometheus
  - Einen Prometheus-Server einrichten, um Metriken der Anwendung zu sammeln.
  - Prometheus so konfigurieren, dass es Metriken vom /actuator/prometheus-Endpunkt derAnwendung abgreift.
  - Optional Alerts einrichten, um Nachrichten zu senden, wenn bestimmte Schwellenwerte überschritten werden, z. B. eine hohe Speichernutzung oder hohe Fehlerraten.
- Grafana
  - Grafana so konfigurieren, dass es sich mit dem Prometheus-Server verbindet und Metriken in Form von Graphen und Diagrammen anzeigt.
  - In Grafana nun benutzerdefinierte Dashboards erstellen, die die für die Anwendung wichtigsten Metriken anzeigen, z. B. CPU-Auslastung, Speichernutzung und Antwortzeiten.

(F) Logging, Tracing, Metrics

1. Logging

Das **Logging** von Vorgängen bezieht sich auf das **Sammeln, Speichern und Analysieren von Protokolldaten**, die von einer Anwendung oder einem System erzeugt werden. Protokolldaten können Informationen wie Anwendungsereignisse, Systemereignisse, Fehler, Warnungen und Leistungskennzahlen enthalten. Die Protokollierung ist ein wichtiger Aspekt des Betriebs, da sie wertvolle Einblicke in das Verhalten und die Leistung einer Anwendung oder eines Systems liefert und zur Fehlersuche und -behebung, zur Ermittlung von Trends und zur Leistungsverbesserung verwendet werden kann.

Es gibt verschiedene Arten von Protokollen, die je nach Anwendung und den Anforderungen des Unternehmens gesammelt werden können:

- **Anwendungsprotokolle** enthalten Informationen über die Ereignisse, die innerhalb einer Anwendung auftreten, z. B. Benutzeraktionen, Datenbankabfragen und Fehlermeldungen.
- **Systemprotokolle** enthalten Informationen über Ereignisse, die innerhalb der zugrunde liegenden Infrastruktur auftreten, z. B. Systemausfälle, Ressourcennutzung und Sicherheitsereignisse.
- **Zugriffsprotokolle** enthalten Informationen über die an eine Anwendung gestellten Anfragen, z. B. die IP-Adresse des Clients, die Anfragemethode und den Antwortstatus.
- **Audit-Protokolle** enthalten Informationen über sicherheitsrelevante Ereignisse, die innerhalb einer Anwendung oder eines Systems auftreten, z. B. Anmeldeversuche von Benutzern und Zugriff auf sensible Daten.

Protokolle können mit verschiedenen Tools gesammelt werden, z. B. mit Protokollanalysatoren, Protokollbibliotheken und speziellen Protokollierungsplattformen. Die aggregierten Daten können in Dashboards visualisiert werden, und es lassen sich Warnmeldungen einrichten, wenn bestimmte Bedingungen erfüllt sind. Sie können auch zur Fehlerbehebung, zur Ermittlung von Trends und zur Planung künftiger Upgrades und Skalierungen verwendet werden.

Die **Protokollierungsstrategie** muss gut definiert sein, einschließlich der Frage, was protokolliert werden soll, wie protokolliert werden soll (Format, auf welchen Log-Level) und wo die Protokolle gespeichert werden sollen, um sicherzustellen, dass die Protokolldaten vollständig, genau und für die Analyse leicht zugänglich sind. Zugleich muss man mit sensiblen Daten wie Benutzerdaten oder Passwörtern sorgfältig umgehen, aus Sicherheitsgründen, aber auch um Datenschutzbestimmungen nicht zu verletzen.

Es gibt viele **Metriken** und **KPIs,** die für die Überwachung verwendet werden können. Welche am nützlichsten sind, hängen von dem zu überwachenden System oder der Anwendung ab. Einige gängige Kategorien der Überwachung sind:

- **Health:** Dazu gehören Metriken, die den Gesamtstatus und das Wohlergehen des Systems anzeigen, wie CPU- und Speichernutzung, Festplattenspeicher und Netzwerkkonnektivität.
- **Durchsatz:** Diese Metriken messen die Leistung des Systems, z. B. Anfragen pro Sekunde, Antwortzeit und Fehlerquote.
- **Domänenspezifische Metriken:** Hierbei handelt es sich um Metriken, die für einen bestimmten Bereich spezifisch sind, z. B. Finanztransaktionen, Kundenbindung oder Website-Traffic.

Es ist üblich, Protokolle und Traces zu überwachen, um das Verhalten des Systems zu verstehen und auf Anomalien oder Fehler aufmerksam zu machen.

2. Tracing und Unterscheide im Vergleich zu Logging

Tracing und **Logging** sind zwei verwandte, aber unterschiedliche Konzepte auf dem Gebiet der Softwareentwicklung und des Softwarebetriebs.

Ein Trace ist eine "Spur einer Anfrage durch das Gesamtsystem" (in einer längeren Zeitspanne), ein Logeintrag ein konkret auftretendes zu protokollierendes Ereignis. Der Hauptunterschied zwischen Tracing und Logging besteht also darin, dass man im Betrieb Tracing für die Verfolgung des Flusses einer Anfrage durch ein verteiltes System heranzieht - Logging hingegen zur Protokollierung von auftretenden Ereignissen:

- **Logging** bezieht sich auf den Prozess der Aufzeichnung von Informationen über die Ausführung einer Anwendung in einem strukturierten Format, normalerweise in einer Datei oder einer Datenbank, zur späteren Analyse und Fehlerbehebung. So erstellte Logs enthalten Informationen über aufgetretene Ereignisse, wie z. B. die Ausführung einer bestimmten Funktion, den Empfang einer Anfrage oder das Auftreten eines Fehlers.
- **Tracing** hingegen ist eine Technik, mit der der Fluss einer Anfrage oder einer Transaktion durch ein verteiltes System verfolgt werden kann. Es ermöglicht Entwicklern und Betriebsteams, den gesamten Fluss einer Anfrage zu sehen, vom Zeitpunkt ihrer Initiierung bis zu ihrer Fertigstellung, einschließlich aller Dienste und Systeme, die sie auf ihrem Weg berührt. Tracing kann dabei helfen, Engpässe, Fehler und andere Probleme zu identifizieren, die durch die Untersuchung von Protokolldateien allein nicht offensichtlich sind.

3. Wann wird aktiv protokolliert, wann passiv überwacht?

Passive Protokollierung und aktive Überwachung sind zwei verschiedene Techniken, die zur Sammlung und Analyse von Daten über eine Anwendung oder ein System verwendet werden können.

- Bei der **passiven Protokollierung** werden die von einer Anwendung oder einem System erzeugten Protokolldaten gesammelt und für eine spätere Analyse gespeichert. Diese Technik wird in der Regel verwendet, um eine breite Palette von Daten über das Verhalten und die Leistung einer Anwendung zu sammeln, einschließlich Fehlern, Leistungsmetriken und Benutzeraktionen. Die passive Protokollierung ist nützlich, um Trends und Muster in den Daten zu erkennen, die auf ein zugrundeliegendes Problem hindeuten könnten, und um Probleme zu beheben.
- **Aktive Überwachung** hingegen bezieht sich auf den Prozess der aktiven Untersuchung einer Anwendung oder eines Systems, um Daten über dessen Leistung, Verfügbarkeit und Zustand zu sammeln. Die aktive Überwachung erfolgt in der Regel durch das gezielte (aktive) Senden von Anfragen an bestimmte Endpunkte innerhalb einer Anwendung oder eines Systems und das Messen der Antwortzeit, des Durchsatzes und anderer Metriken. Diese Technik ist nützlich für die Überwachung der Echtzeit-Performance einer Anwendung und für die Identifizierung und Behebung von Problemen, die sich auf die Verfügbarkeit oder Performance auswirken können. Außerdem lässt sich auf diese Weise überprüfen, ob ein bestimmter Dienst, eine Komponente oder ein Endpunkt wie erwartet funktioniert.

5. Was bedeutet die Verwendung solcher Metriken und Logging- und Tracing-Tools für das Design einer Anwendung?

Der Einsatz von Metriken, Logging- und Tracing-Tools kann sich erheblich auf das Design einer Anwendung auswirken, da die Anwendung so instrumentiert werden muss, dass Daten aus verschiedenen Teilen des Systems erfasst werden können.

- **Instrumentierung:** Die Anwendung muss so konzipiert sein, dass sie Metriken, Protokolle und Traces auf organisierte Weise ausgeben kann. Dazu muss im Code dafür gesorgt werden, dass diese Ereignisse an den entsprechenden Stellen geschrieben werden.
- **Datenerfassung** und **-speicherung:** Die Anwendung sollte so konzipiert sein, dass eine effiziente Erfassung und Speicherung der von der Instrumentierung ausgegebenen Daten möglich ist. Dazu kann das Senden von Daten an Überwachungswerkzeuge, Protokollierungssysteme oder Datenspeicherlösungen gehören.
- **Datenanalyse:** Die Anwendung sollte so konzipiert sein, dass eine einfache Analyse der erfassten Daten möglich ist, z. B. durch die Verwendung von Dashboards oder Abfragesprachen. Eine Möglichkeit ist beispielsweise die Verwendung von Request-IDs, um eine Anfrage in den Logs identifiziereung und durch eine Anwendung hindurch verfolgen zu können.
- Integration mit **Alarmierungssystemen:** Die Anwendung sollte auch so konzipiert sein, dass sie in Warnsysteme integriert werden kann, so dass alle erkannten Probleme schnell an die zuständigen Stellen weitergeleitet werden können.

6. Auswirkungen und Folgen, Vor- und Nachteile, für das Laufzeitverhalten der Anwendung beim Einsatz solcher Techniken

Der Einsatz von **Metriken, Logging- und Tracing-Techniken** in einer Anwendung hat sowohl Vorteile als auch Nachteile.

Vorteile:

- Ermöglicht die Überwachung des Systemzustands und der Systemleistung in Echtzeit
- Ermöglicht die schnellere Identifizierung und Lösung von Problemen
- Bietet wertvolle Einblicke in das Verhalten des Systems im Laufe der Zeit

Nachteile:

- Kann die Anwendung hinsichtlich der Ressourcennutzung (z. B. CPU und Speicher) zusätzlich belasten Um die Leistung hoch zu halten, ist es wichtig, den Overhead bei der Überwachung und Protokollierung zu minimieren. Die Logdaten und -frequenz ist sorgfältig zu wählen, effizi ente Lösungen für die Datenerfassung und -speicherung müssen verwendet werden. Idealerweise werden Log-Informationen asynchron weggeschrieben.
- Kann den Code der Anwendung komplexer machen
- Kann Sicherheitsrisiken mit sich bringen, wenn sensible Daten protokolliert oder nachverfolgt werden Daher ist es wichtig, sicherzustellen, dass sensible Daten wie Passwörter nicht protokolliert oder nachverfolgt werden. Logs sollte nur mit geeigneter Authentifizierung /Autorisierung zugreifbar sein.
- Darüber hinaus ist es wichtig, einen Plan für die Verwaltung der erfassten Daten zu haben, einschließlich Aufbewahrungsrichtlinien und Archivierungsstrategien, um sicherzustellen, dass die Daten nur so lange aufbewahrt werden, wie sie benötigt werden.

7. Wichtige Systemmetriken für Zustands- und Performance-Überwachung von Anwendungen

Bei der **Überwachung von Zustands und Performance einer Java- und Spring-Boot-basierten Anwendung** gibt es mehrere wichtige Systemmetriken, die berücksichtigt werden sollten:

1. **JVM-Metriken:** Dazu gehören Metriken wie die Heap- und Non-Heap-Speicherauslastung, GC-Aktivität (Garbage Collection) und Thread-Anzahl. Diese Metriken geben Aufschluss über die Speichernutzung und die Leistung der JVM.
2. **CPU-Nutzung:** Diese Metrik gibt Aufschluss über die CPU-Ressourcen, die die Anwendung verbraucht. Eine hohe CPU-Auslastung kann darauf hindeuten, dass die Anwendung eine große Menge an Berechnungen durchführt oder dass sie auf Leistungsengpässe stößt.
3. **Speichernutzung:** Diese Metrik gibt Aufschluss über die von der Anwendung beanspruchte Speichermenge.

4. **Netzwerkaktivität:** Diese Metrik gibt Aufschluss über den Umfang des Netzwerkverkehrs, den die Anwendung erzeugt. Eine hohe Netzwerkaktivität kann darauf hinweisen, dass die Anwendung mit einer großen Anzahl externer Dienste kommuniziert oder dass sie eine große Menge an Netzwerkressourcen verbraucht.
5. **Anfrage/Antwort-Metriken:** Dazu gehören Metriken wie Anforderungsrate, Fehlerrate und Antwortzeit. Diese Metriken geben Aufschluss über die Leistung der Anwendung bei der Verarbeitung eingehender Anfragen.

8. DORA Metriken

Info
https://www.leanix.net/en/wiki/vsm/dora-metrics

DORA steht für "**The DevOps Research and Assessment Team**". Als Google-Forschungsgruppe analysierte es die Leistung von DevOps-Teams bei Softwareentwicklung und -bereitstellung mit diesen Metriken:

- Deployment-Frequenz: Bezieht sich auf die Häufigkeit erfolgreicher Software-Releases für die Produktion.
- Vorlaufzeit für Änderungen: Erfasst die Zeit zwischen dem Commit einer Code-Änderung und ihrem einsatzfähigen Zustand.
- MTTR: Misst die Zeit zwischen einer Unterbrechung aufgrund eines Deployments oder eines Systemausfalls und der vollständigen Wiederherstellung. Siehe Kapitel 8.
- Change Failure Rate: Gibt an, wie oft die Änderungen oder Hotfixes eines Teams zu Fehlern führen, nachdem der Code bereitgestellt wurde.

9. Sammeln und Korrelieren der Log-Ausgaben: Empfohlene Tools für Spring-Boot-Anwendungen

Es gibt verschiedene Möglichkeiten, die **Logausgaben von Diensten zu sammeln und zu korrelieren**, je nach den spezifischen Anforderungen Ihrer Anwendung und Infrastruktur. Einige gängige Ansätze sind:

1. **Zentralisiertes Logging:** Bei diesem Ansatz werden alle Logmeldungen von verschiedenen Diensten an einen zentralen Server gesendet, wo sie gesammelt, geparst und analysiert werden können. Tools wie Elasticsearch, Logstash und Kibana **(ELK-Stack)** oder **Graylog** sind eine gängige Wahl für das zentralisierte Logging.
2. **Verteiltes Tr**acing: Bei diesem Ansatz wird eine Anfrage verfolgt, während sie durch verschiedene Dienste fließt. Informationen über die Anfrage und die Antwort werden in einem zentralen Speicher gesammelt. Tools wie **Zipkin,** Jaeger und Appdash sind eine gängige Wahl für verteiltes Tracing.
3. **Log-Aggregation:** Bei diesem Ansatz wird ein Log-Aggregator-Tool verwendet, um Log-Daten aus mehreren Quellen zu sammeln und eine einheitliche Ansicht zu bieten. Log-Aggregatoren wie Fluentd, Logagent und Logstash können die Logs parsen und an einen zentralen Speicher, z. B. Elasticsearch, weiterleiten.

Für Spring Boot-Anwendungen empfieht sich Spring Cloud Sleuth und Zipkin. Spring Cloud Sleuth ist ein Open-Source-Framework, das verteiltes Tracing in Spring Boot-Anwendungen ermöglicht. Zipkin ist ein verteiltes Tracing-System, mit dem man Traces von Services sammeln, durchsuchen und visualisieren kann.

Außerdem bietet Spring Boot standardmäßig Unterstützung für verschiedene Logging-Frameworks wie logback und log4j2, um die Logs zu schreiben.

10. Verbreitete Tools für die zentrale Verwaltung von Protokolldaten

1. Elasticsearch, Logstash und Kibana **(ELK-Stack):** Dieser Open-Source-Stack wird häufig für Log-Management verwendet, da er die Sammlung, Speicherung und Visualisierung von Protokolldaten ermöglicht.
2. **Splunk:** Hierbei handelt es sich um ein kommerzielles Tool zur Log-Management, das leistungsstarke Indizierungs- und Suchfunktionen sowie die Möglichkeit bietet, benutzerdefinierte Dashboards und Warnmeldungen zu erstellen.
3. **Fluentd:** Hierbei handelt es sich um einen Open-Source-Tool, mit dem Protokolldaten gesammelt und an einen zentralen Ort zur Speicherung und Analyse weitergeleitet werden können.
4. **Loggly:** Hierbei handelt es sich um einen Cloud-basierten Log-Management-Service, der die Sammlung, Speicherung und Analyse von Protokolldaten aus verschiedenen Quellen ermöglicht.
5. **Graylog** ist ein Open-Source-Tool für das Log-Management, das die Sammlung, Speicherung und Analyse von Protokolldaten aus verschiedenen Quellen ermöglicht und über eine Webschnittstelle für die Suche und Alarmierung verfügt.
6. **Papertrail** ist ein Cloud-basierten Log-Management-Service, der das Sammeln, Speichern und Analysieren von Protokolldaten aus verschiedenen Quellen mit einer Webschnittstelle für die Suche und Alarmierung ermöglicht.

11. Verbreitete Tools für die zentrale Verwaltung von Metriken

1. **Prometheus:** Hierbei handelt es sich um ein Open-Source-System zur Sammlung und Speicherung von Metriken, das in Cloud-nativen Umgebungen weit verbreitet ist. Es bietet eine leistungsstarke Abfragesprache und die Möglichkeit, benutzerdefinierte Warnmeldungen zu erstellen.
2. **InfluxDB:** Hierbei handelt es sich um eine Open-Source-Zeitreihendatenbank, die häufig zum Speichern und Abfragen von Metrikdaten verwendet wird. Sie enthält auch eine integrierte Abfragesprache und Warnfunktionen.
3. **Grafana:** Hierbei handelt es sich um ein Open-Source-Tool zur Visualisierung von Metriken, das zur Erstellung von benutzerdefinierten Dashboards und Warnmeldungen verwendet werden kann. Es unterstützt eine breite Palette von Datenquellen, einschließlich Prometheus und InfluxDB.

4. **Datadog:** Hierbei handelt es sich um eine Cloud-basierte Plattform für die Verwaltung und Überwachung von Metriken, die die Sammlung, Speicherung und Analyse von Metrikdaten aus verschiedenen Quellen ermöglicht. Sie verfügt außerdem über integrierte Funktionen zur Alarmierung und Erkennung von Anomalien.
5. **New Relic:** Dies ist eine Cloud-basierte Plattform zur Leistungsüberwachung und -analyse, die Echtzeiteinblicke in die Leistung einer Anwendungen, Infrastruktur und Protokolle bietet.
6. **Zabbix:** Hierbei handelt es sich um eine Open-Source-Überwachungslösung, die die Erfassung, Speicherung und Analyse von Metrikdaten ermöglicht. Sie umfasst auch eine integrierte Abfragesprache und Warnfunktionen.
7. **Nagios:** Hierbei handelt es sich um eine Open-Source-Überwachungslösung, die die Überwachung von Netzwerkdiensten und Hosts ermöglicht. Sie umfasst Warnmeldungen und Berichtsfunktionen.

12. Logging-Frameworks für Java- und Spring-Boot-Anwendungen

1. **Log4j:** Log4j ist ein Java-basiertes Logging-Framework, das Teil des Apache Logging Services Project ist. Es ist weit verbreitet, flexibel und gut konfigurierbar.
2. **Logback:** Logback ist ein Logging-Framework für Java, das als Nachfolger des beliebten log4j-Frameworks gedacht ist. Es wurde entwickelt, um schneller, zuverlässiger und flexibler als log4j zu sein.
3. Java Util Logging **(JUL):** JUL ist ein in Java eingebautes Logging-Framework, das einfach zu benutzen ist und grundlegende Logging-Funktionen bietet.
4. **SLF4J:** SLF4J ist eine Fassade, bzw. Abstraktion für verschiedene Logging-Frameworks. Es ermöglicht, das gewünschte Logging-Framework erst zur Laufzeit einzubinden.
5. Der **Standard-Logger** von Spring Boot: Spring Boot verwendet Commons Logging für sein internes Logging, kann aber leicht so konfiguriert werden, dass jedes andere der oben genannten Logging-Frameworks verwendet wird.

(G) Operations Database

Info
Michael T. Nygard: "Release It!: Design and Deploy Production-Ready Software", Pragmatic Programmers, 2017

1. Operations Database (Michael Nygard)

Die **Operations Database (OpsDB)** ist ein Konzept, das von Michael Nygard in seinem Buch "*Release It! Design and Deploy Production-Ready Software*" vorschlägt. Es handelt sich um eine Datenbank, in der Betriebsdaten wie Metriken, Protokollierung und Konfiguration gespeichert werden, die von verschiedenen Teilen des Systems gesammelt werden.

Der Hauptzweck der OpsDB besteht darin, einen zentralen Ort für die Speicherung und Analyse von Betriebsdaten bereitzustellen, die dann zur Überwachung des Zustands und der Leistung des Systems, zur Fehlerbehebung und für datengestützte Entscheidungen verwendet werden können.

Die OpsDB ist so konzipiert, dass sie hochverfügbar und fehlertolerant ist und hohe Schreib- und Leselasten bewältigen kann. Sie ist außerdem so konzipiert, dass sie mit Tools wie SQL oder einer speziellen Abfragesprache leicht abgefragt und analysiert werden kann.

Einer der Hauptvorteile der OpsDB besteht darin, dass sie eine **Trennung der operativen Daten von den eigentlichen Geschäftsdaten** ermöglicht, was die Verwaltung und Analyse der Daten erleichtert. Dies ermöglicht auch ein klares Separation of Concerns zwischen Anwendungs- und Betriebsteam. Eine OpsDB ist somit kein Ersatz für Unternehmensdatenbanken, sondern vielmehr ein ergänzendes Tool für die Speicherung und Analyse von Betriebsdaten.

Die OpsDB ermöglicht auch eine bessere Beobachtbarkeit und Überwachung des Systems, da sie eine einzige Quelle der Wahrheit für die Betriebsdaten bietet. Dies ermöglicht eine effizientere Fehlerbehebung und Ursachenanalyse sowie die Erstellung von benutzerdefinierten Metriken und Warnmeldungen auf der Grundlage der in der OpsDB gespeicherten Daten.

2. Analyse-Tracing und Metrik-Tools von der Applikation trennen

Es ist aus mehreren Gründen eine gute Idee, **Analyse-, Tracing- und Metrik-Tools aus der Anwendung** selbst herauszuhalten und sie auf verschiedene Systeme aufzuteilen:

1. **Entkopplung:** Die Trennung der Überwachungs- und Protokollierungsfunktionen von der Anwendung ermöglicht eine bessere Entkopplung von Belangen, was die Wartung und Aktualisierung der Anwendung erleichtert. Außerdem kann die Überwachungs- und Protokollierungsfunktionalität unabhängig von der Anwendung weiterentwickelt werden, was für das Hinzufügen neuer Funktionen oder die Behebung von Fehlern nützlich sein kann.
2. **Performance:** Die Trennung der Überwachungs- und Protokollierungsfunktionen von der Anwendung kann dazu beitragen, den Overhead zu verringern, den die Überwachung und Protokollierung auf die Leistung der Anwendung haben kann. Dies liegt daran, dass die Überwachungs- und Protokollierungsfunktionalität unabhängig von der Anwendung optimiert und skaliert werden kann.
3. **Skalierbarkeit:** Die Trennung der Überwachungs- und Protokollierungsfunktionen von der Anwendung kann zu einer besseren Skalierbarkeit beitragen. Durch den Einsatz spezialisierter Tools, wie z. B. eines Log-Aggregators oder eines Metriksammlers, ist es möglich, große Datenmengen zu verarbeiten und das System entsprechend den sich ändernden Anforderungen zu skalieren.
4. **Flexibilität:** Die Trennung der Überwachungs- und Protokollierungsfunktionen von der Anwendung kann die Flexibilität verbessern. Durch den Einsatz spezialisierter Tools ist es einfacher, das beste Tool für die jeweilige Aufgabe auszuwählen und zwischen verschiedenen Tools zu wechseln, wenn sich die Anforderungen des Systems ändern.
5. **Sicherheit:** Die Trennung der Überwachungs- und Protokollierungsfunktionen von der Anwendung kann zur Verbesserung der Sicherheit beitragen, da sie getrennt von der Anwendung gesichert werden können. Außerdem wird die Einhaltung von Vorschriften erleichtert, da die Daten gemäß den gesetzlichen Bestimmungen erfasst, gespeichert und analysiert werden können.
6. **Nachvollziehbarkeit:** Die Trennung der Überwachungs- und Protokollierungsfunktionen von der Anwendung ermöglicht ein klares Verständnis des Systemzustands und der daran vorgenommenen Änderungen, was die Nachvollziehbarkeit verbessern kann.

Siehe dazu Kapitel 6 (A) zum Einsatz von AOP und Interceptoren.

(H) Cloud

1. Cloud-Native Design, typische Stacks und Patterns

Cloud-natives Design für eine Anwendung bezieht sich auf die Grundsätze und Praktiken für die Erstellung, Bereitstellung und Ausführung von Anwendungen in einer Cloud-Umgebung.

Das Hauptziel von Cloud-nativem Design besteht darin, die Skalierbarkeit, Verfügbarkeit und andere Funktionen von Cloud-Computing-Plattformen zu nutzen, um Anwendungen zu erstellen, die stabiler, leistungsfähiger und kostengünstiger sind.

Zu den wichtigsten Grundsätzen des **Cloud-nativen Designs** gehören:

- Microservices-Architektur: Kleine, unabhängig voneinander einsetzbare Dienste ermöglichen eine schnellere Entwicklung, eine effizientere Skalierung und eine bessere Fehlerisolierung.
- Containerisierung: Hierbei werden eine Anwendung und ihre Abhängigkeiten in einen Container verpackt, der dann in einer Cloud-Umgebung bereitgestellt und ausgeführt werden kann. Container bieten eine konsistente und isolierte Laufzeitumgebung für die Anwendung.
- Automatisierung und Orchestrierung: Dies beinhaltet die Automatisierung der Bereitstellung, Skalierung und Verwaltung der Anwendung und ihrer Abhängigkeiten.
- Immutable Infrastructure: Dabei wird die Infrastruktur, auf der eine Anwendung läuft, als unveränderlich behandelt, d. h. sie sollte eher ersetzt als verändert werden. Siehe Kapitel 5 (D) 4.
- Selbstheilung und Selbstoptimierung: Hier geht es um die Entwicklung von Anwendungen, die Fehler automatisch erkennen und beheben sowie ihre eigene Leistung automatisch optimieren können.

Typische **Stacks** für Cloud-native Anwendungen sind (Auswahl, vgl. auch andere Abschnitte z.B. in Kapitel 5 oder 8):

- Kubernetes, Docker und Prometheus für die Orchestrierung und Überwachung von Containern.
- Spring Boot als Anwendungsrahmen für die Services
- Consul, Eureka, Zookeeper für Service Discovery
- Istio, Envoy als Service Mesh
- Grafana, Kibana für die Visualisierung und Analytik
- Jenkins, Travis für CI/CD-Pipelines

2. Betriebstools, die in Cloud-Umgebungen wie AWS, Azure und Google Cloud verwendet werden

Cloud-Anbieter wie AWS, Azure und Google Cloud bieten eine breite Palette von **Tools für den Betrieb und die Verwaltung von Cloud-basierten Anwendungen und Infrastrukturen**. Einige der gängigen Tools sind:

- CloudFormation, Terraform und ARM Templates: Hierbei handelt es sich um Infrastructure-as-Code-Tools (IaC), mit denen Benutzer Cloud-Ressourcen mithilfe von Code bereitstellen und verwalten können, anstatt sie manuell zu konfigurieren.
- CloudWatch, Azure Monitor und Stackdriver sind Cloud-native Überwachungs- und Protokollierungslösungen, mit denen Benutzer Metriken und Protokolle von Cloud-basierten Ressourcen sammeln, speichern und analysieren können.
- Elastic Beanstalk, App Service und App Engine: Hierbei handelt es sich um Platform-as-a-Service (PaaS)-Lösungen, mit denen Benutzer Anwendungen bereitstellen und ausführen können, ohne die zugrunde liegende Infrastruktur verwalten zu müssen.
- EC2 Auto Scaling, Virtual Machine Scale Sets und Kubernetes Engine sind Lösungen zur automatischen Skalierung der Anzahl der Instanzen einer Anwendung je nach Bedarf.
- Elastic Load Balancing, Azure Load Balancer und Cloud Load Balancing sind Lösungen für die Verteilung des eingehenden Datenverkehrs auf mehrere Instanzen einer Anwendung.
- S3, Azure Blob Storage und Google Cloud Storage sind Object Stores, die es den Benutzern ermöglichen, große Datenmengen zu speichern und abzurufen.
- RDS, Azure SQL Database und Cloud SQL: Hierbei handelt es sich um relationale Datenbankdienste, mit denen Nutzer eine relationale Datenbank in der Cloud einfach bereitstellen, verwalten und skalieren können.
- CloudFront, Azure CDN und Cloud CDN: Hierbei handelt es sich um Content Delivery Network (CDN)-Services, mit denen Benutzer Inhalte über mehrere Standorte verteilen können, um sie schneller bereitzustellen.
- AWS Lambda, Azure Functions und Cloud Functions: Hierbei handelt es sich um Serverless-Computing-Lösungen, die es Benutzern ermöglichen, Code auszuführen, ohne Server bereitstellen oder verwalten zu müssen.

3. Cloud und IaaS

Cloud Infrastructure as a Service (IaaS) ist eine Kategorie des Cloud Computing, die Nutzern über das Internet Zugang zu rohen Rechenressourcen wie virtuellen Maschinen, Speicher und Netzwerken bietet. IaaS-Anbieter wie Amazon Web Services (AWS), Microsoft Azure und Google Cloud Platform (GCP) ermöglichen es den Nutzern, diese Ressourcen nach Bedarf zu mieten, ohne dass sie in eine eigene physische Infrastruktur investieren und diese verwalten müssen.

Mit IaaS kann man Rechenressourcen je nach Bedarf schnell und einfach vergrößern oder verkleinern. Iaas kann für eine breite Palette von Anwendungen verwendet werden, einschließlich Web- und Mobilanwendungen, Big Data-Verarbeitung und maschinelles Lernen.

IaaS-Anbieter bieten in der Regel eine breite Palette von Dienstleistungen an, darunter:

- Virtuelle Maschinen: Benutzer können virtuelle Maschinen erstellen und verwalten, auf denen verschiedene Betriebssysteme wie Windows, Linux und macOS ausgeführt werden können.
- Speicher: Benutzer können Daten in der Cloud speichern und darauf zugreifen, einschließlich Object Stores, Blockspeicher und Dateispeicher.
- Vernetzung: Benutzer können virtuelle Netzwerke erstellen und verwalten, einschließlich virtueller privater Clouds (VPCs), Lastverteiler und VPNs.
- Datenbanken: Benutzer können relationale und nicht-relationale Datenbanken erstellen und verwalten, darunter MySQL, SQL Server und MongoDB.
- weitere Dienste wie Sicherheit, Überwachung und Automatisierung

4. IaaS, PaaS, SaaS

IaaS, PaaS und SaaS sind drei Kategorien von Cloud Computing-Diensten, die sich durch die Abstraktionsebene unterscheiden, die sie den Benutzern bieten.

- **IaaS** (Infrastructure as a Service) ist die grundlegendste Stufe des Cloud Computing und bietet den Nutzern Zugang zu Rohdatenverarbeitungsressourcen wie virtuellen Maschinen, Speicher und Netzwerken. Beispiele für IaaS-Anbieter sind Amazon Web Services (AWS), Microsoft Azure und Google Cloud Platform (GCP).
- **PaaS** (Platform as a Service) baut auf IaaS auf und bietet Benutzern eine Plattform, auf der sie ihre Anwendungen entwickeln, ausführen und verwalten können. PaaS-Anbieter bieten in der Regel diverse Diensten wie Datenbanken, Webserver und Entwicklungs-Frameworks an, die zur Erstellung und Ausführung von Anwendungen verwendet werden können. Beispiele für PaaS-Anbieter sind Heroku, Google App Engine und AWS Elastic Beanstalk.
- **SaaS** (Software as a Service) ist die höchste Abstraktionsebene des Cloud Computing und bietet den Benutzern Zugang zu Softwareanwendungen, die vom Anbieter gehostet und verwaltet werden. Die Nutzer greifen über einen Webbrowser oder eine mobile Anwendung auf diese Anwendungen zu und müssen sich nicht um die zugrunde liegende Infrastruktur oder Plattform kümmern. Beispiele für SaaS-Anbieter sind Salesforce, Microsoft Office 365 und Google G Suite.