Curriculum für

Certified Professional for Software Architecture (CPSA)®
Advanced Level

Modul
FLEX

Flexible Architekturmodelle - Modularisierung, Integration und Betrieb von modernen Softwaresystemen

2024.1-rev0-DE-20241231

Inhaltsverzeichnis

Die Abkürzung "e. V." ist Teil des offiziellen Namens des iSAQB und steht für "eingetragener Verein", der seinen Status als juristische Person nach deutschem Recht beschreibt. Der Einfachheit halber wird iSAQB e. V. im Folgenden ohne die Verwendung dieser Abkürzung als iSAQB bezeichnet.

Verzeichnis der Lernziele

- LZ 6-4: [OPTIONAL] Service Level Objectives (SLOs) aus Qualitätszielen ableiten
- LZ 6-5: [OPTIONAL] Mit Architektur das Incident Management und schnelle MTTR unterstützen
- LZ 6-6: [OPTIONAL] Durch Architektur zu Disaster Recovery und Business Continuity Management beitragen
- LZ 6-7: [OPTIONAL] Chaos Engineering anhand von Qualitätszielen designen und durchführen

Einführung: Allgemeines zum iSAQB Advanced Level

Was vermittelt ein Advanced Level Modul?

Das Modul kann unabhängig von einer CPSA-F-Zertifizierung besucht werden.

- Der iSAQB Advanced Level bietet eine modulare Ausbildung in drei Kompetenzbereichen mit flexibel gestaltbaren Ausbildungswegen. Er berücksichtigt individuelle Neigungen und Schwerpunkte.
- Die Zertifizierung erfolgt als Hausarbeit. Die Bewertung und mündliche Prüfung wird durch vom iSAQB benannte Experten vorgenommen.

Was können Absolventen des Advanced Level (CPSA-A)?

CPSA-A-Absolventen können:

- eigenständig und methodisch fundiert mittlere bis große IT-Systeme entwerfen
- in IT-Systemen mittlerer bis hoher Kritikalität technische und inhaltliche Verantwortung übernehmen
- Maßnahmen zur Erreichung von Qualitätsanforderungen konzeptionieren, entwerfen und dokumentieren sowie Entwicklungsteams bei der Umsetzung dieser Maßnahmen begleiten
- architekturrelevante Kommunikation in mittleren bis großen Entwicklungsteams steuern und durchführen

Voraussetzungen zur CPSA-A-Zertifizierung

- erfolgreiche Ausbildung und Zertifizierung zum Certified Professional for Software Architecture, Foundation Level® (CPSA-F)
- mindestens drei Jahre Vollzeit-Berufserfahrung in der IT-Branche; dabei Mitarbeit an Entwurf und Entwicklung von mindestens zwei unterschiedlichen IT-Systemen
    - Ausnahmen sind auf Antrag zulässig (etwa: Mitarbeit in Open-Source-Projekten)
- Aus- und Weiterbildung im Rahmen von iSAQB-Advanced-Level-Schulungen im Umfang von mindestens 70 Credit-Points aus mindestens drei unterschiedlichen Kompetenzbereichen
- erfolgreiche Bearbeitung der CPSA-A-Zertifizierungsprüfung

Grundlegendes

Was vermittelt das Modul „FLEX“?

Das Modul Flexible Architekturmodelle - Modularisierung, Integration und Betrieb von modernen Softwaresystemen zeigt, wie man mit modernen Architektur-Ansätzen besonders flexible Systeme umsetzen kann. Hierbei beginnen wir bei Qualitätszielen, die flexible, verteilte Systeme überhaupt motivieren, betrachten Modularisierungs-Optionen und Architektur-Konzepte. Die Teilnehmer:innen lernen wie Kommunikationsstrukturen zwischen Teams und deren Organisation ihre Architekturentscheidungen entscheidend beeinflussen und wie sie diese Macro-Architektur steuern können.

Am Ende des Moduls kennen die Teilnehmer:innen Grundlagen von Microservices und Self-contained Systems und können Systeme auf Basis dieser Architektur-Konzepte entwerfen. Dabei können sie auch eine sinnvolle fachliche Aufteilung vorschlagen, die Erkenntnisse aus Konzepten wie Continuous Delivery und nachhaltigem Service-Betrieb berücksichtigt.

Struktur des Lehrplans und empfohlene zeitliche Aufteilung

Inhalt	Empfohlene Mindestdauer (min)	Übungszeit (Minuten)
1. Warum flexible Systeme einsetzen	60	30
2. Modularisierung von Systemen in Teilsysteme	120	60
3. Softwaremodule und der Bezug zu Organisationsstrukturen	90	60
4. Integrationsmethoden & Protokolle	150	60
5. Installation und Laufzeitumgebungen/ Plattformen	90	90
6. Betriebsmodelle	90	60
Summe	600 (10h)	360 (6h)

In diesem Kontext stellt "Mindestdauer" die Unterrichtszeit ohne Übungszeit dar. Die als OPTIONAL markierten Inhalte und Lernziele kann jeder Trainer hinzunehmen oder weglassen, sollte dies aber in der Trainingsbeschreibung den Teilnehmern vorzeitig bekannt machen. Die Zeitaufteilung zwischen Theorie und Übungen ist lediglich eine Empfehlung. Die Ausgestaltung der Beispiele und Übungen sind in diesem Lehrplan nicht vorgegeben.

Dauer, Didaktik und weitere Details

Die unten genannten Zeiten sind Empfehlungen. Die Dauer einer Schulung zum Modul FLEX sollte mindestens 3 Tage betragen, kann aber länger sein. Anbieter können sich durch Dauer, Didaktik, Art und Aufbau der Übungen sowie der detaillierten Kursgliederung voneinander unterscheiden. Insbesondere die Art der Beispiele und Übungen lässt der Lehrplan komplett offen.

Lizenzierte Schulungen zu FLEX tragen zur Zulassung zur abschließenden Advanced-Level-Zertifizierungsprüfung folgende Credit Points bei:

Methodische Kompetenz:	10 Punkte
Technische Kompetenz:	20 Punkte
Kommunikative Kompetenz:	0 Punkte

Voraussetzungen

Teilnehmerinnen und Teilnehmer **sollten** folgende Kenntnisse und/oder Erfahrung mitbringen:

- Grundlagen der Beschreibung von Architekturen mithilfe verschiedener Sichten, übergreifender Konzepte, Entwurfsentscheidungen, Randbedingungen etc., wie es im CPSA-F (Foundation Level) vermittelt wird.
- Erfahrung mit der Implementierung und Architektur in agilen Projekten.
- Erfahrungen aus der Entwicklung und Architektur klassischer Systeme mit den typischen Herausforderungen.

Hilfreich für das Verständnis einiger Konzepte sind darüber hinaus:

- Verteilte Systeme
  - Probleme und Herausforderungen bei der Implementierung verteilter Systeme
  - Typische verteilte Algorithmen
  - Internet-Protokolle
- Kenntnisse über Modularisierungen
  - Fachliche Modularisierung
  - Technische Umsetzungen wie Pakete oder Bibliotheken
- Klassische Betriebs- und Deployment-Prozesse

Gliederung des Lehrplans

Die einzelnen Abschnitte des Lehrplans sind gemäß folgender Gliederung beschrieben:

- **Begriffe/Konzepte**: wesentliche Kernbegriffe dieses Themas.
- **Unterrichts-/Übungszeit**: Legt die Unterrichts- und Übungszeit fest, die für dieses Thema bzw. dessen Übung in einer akkreditierten Schulung mindestens aufgewendet werden muss.
- **Lernziele**: Beschreibt die zu vermittelnden Inhalte inklusive ihrer Kernbegriffe und -konzepte.

Dieser Abschnitt skizziert damit auch die zu erwerbenden Kenntnisse in entsprechenden Schulungen. Der Aufbau der Kapitel folgt einer klaren Progression:

- Erst das große Bild verstehen
- Dann die Voraussetzungen kennen
- Die organisatorischen Aspekte durchdringen
- Die konkreten Architekturentscheidungen vorbereiten und bewerten
- Die langfristigen Ziele im Blick haben
- Und schließlich: Entscheidungen fundiert argumentieren und erklären können

Ergänzende Informationen, Begriffe, Übersetzungen

Soweit für das Verständnis des Lehrplans erforderlich, haben wir Fachbegriffe ins iSAQB-Glossar aufgenommen, definiert und bei Bedarf durch die Übersetzungen der Originalliteratur ergänzt.

1. Warum flexible Systeme einsetzen

Dauer: 60 Min.	Übungszeit: 30 Min.

1.1. Begriffe und Konzepte

Verfügbarkeit, Zuverlässigkeit, Time-to-Market, Flexibilität, Vorhersagbarkeit, Reproduzierbarkeit, Internet/Web-Scale, verteilte Systeme, Parallelisierbarkeit der Feature-Entwicklung, Evolution der Architektur (Build for Replacement), Heterogenität, Automatisierbarkeit.

1.2. Lernziele

LZ 1-1: Themen und Buzzwords einordnen

- Erklären, wie sich Architekturstile wie Microservices, Self-contained Systems, Modulithen und Monolithen unterscheiden
- Die Rolle von DevOps und Continuous Delivery im Kontext flexibler Architekturen verstehen
- Den Zusammenhang zwischen Cloud Computing und flexiblen Architekturen erkennen
- Aktuelle Architektur-Trends anhand gegebener Qualitätsziele einschätzen und bewerten

LZ 1-2: Voraussetzungen für verteilte Systeme verstehen und analysieren

- Die wichtigsten technischen Voraussetzungen für verteilte Systeme identifizieren
- Die Bedeutung von Architektur für schnelle Auslieferung von Features erkennen
- Verstehen, wie Team-Abhängigkeiten die Entwicklungsgeschwindigkeit beeinflussen
- Erkennen, welche Team-Skills für verteilte Systeme benötigt werden
- Die Rolle einheitlicher Entwicklungsumgebungen für Fehlerreduktion und -reproduzierbarkeit verstehen
- Die Beziehung zwischen CI, CD und Architektur erkennen
- Verstehen, warum Automatisierung, Wiederholbarkeit und Resilienz für verteilte Systeme wichtig ist
- Die Vorteile verschiedener Isolationsarten (Devtime, Runtime, Deployment, Team) bewerten

LZ 1-3: Kompromisse der vorgestellten Architekturtypen vermitteln und adaptieren

- Die Vor- und Nachteile verschiedener Architekturstile abwägen
- Wissen, wie man Architekturentscheidungen dem Business-Kontext anpasst
- Vor- und Nachteile von Remotekommunikation verstehen und als Entscheidungskriterium für oder gegen Verteilung nutzen
- Die verschiedenen Isolationsarten kennen um die passende Balance zwischen Isolation und Komplexität finden
- Verstehen, wann die Kombination verschiedener Architekturstile sinnvoll ist
- Die Kosten von Verteilung und Service-Kommunikation realistisch einschätzen

LZ 1-4: Langfristige Qualitätsziele von flexiblen Architekturen benennen

- Zentrale Qualitätsmerkmale flexibler Architekturen kennen

- Verstehen, warum Flexibilität und schnelles Feedback strategische Ziele sind
- Erklären, dass die Balance zwischen kurzfristiger und langfristiger Optimierung nicht in jedem Fall ausgewogen ist
- Methoden zur Messbarkeit von Qualitätszielen erarbeiten
- Langfristige Ziele in schrittweise Verbesserungen herunterbrechen und Zusammenhänge darstellen
- Die Rolle von Reproduzierbarkeit und Vorhersagbarkeit verstehen
- Automatisierbarkeit als Qualitätsziel erkennen

LZ 1-5: Typische Architekturentscheidungen von flexiblen Architekturen erklären und begründen

- Architekturentscheidungen durch Businessziele und Qualitätsszenarien begründen
- Die Notwendigkeit bestimmter Architekturmuster erklären
- Kosten von Architekturentscheidungen transparent machen
- Den Wert von architektonischer Flexibilität vermitteln
- Entscheidungen für oder gegen Remotekommunikation begründen
- Die Wahl von Isolationsgrenzen rechtfertigen

1.3. Referenzen

[Wolff 2016], [Humble, et al. 2014], [Lewis, Fowler, et al. 2013], [Takada 2013]

2. Modularisierung von Systemen in Teilsysteme

Dauer: 120 Min.	Übungszeit: 60 Min.

2.1. Begriffe und Konzepte

- Motivation für die Dekomposition in kleinere Systeme
- Unterschiedliche Arten von Modularisierung, Kopplung
- Systemgrenzen als Mittel für Isolation
- Hierarchische Struktur
- Anwendung, Applikation, Self-contained System, Microservice
- Domain-Driven Design-Konzepte und „Strategic Design“, Bounded Contexts

2.2. Lernziele

LZ 2-1: Dekomposition in Bausteine anhand der Anforderungen entwerfen

- Zerlegungsansätze: Erstellung einer Systemzerlegung in Bausteine unter Berücksichtigung fachlicher und technischer Anforderungen.
- Strategic Design: Einsatz von Domain-Driven Design (DDD) zur Definition von Modulgrenzen anhand fachlicher Domänen (z. B. Bounded Contexts).
- Hierarchische Strukturen: Berücksichtigung der Hierarchie von Modulen bei der Zerlegung, z. B. Subdomänen und Context-Mapping.
- Namen mit eindeutiger Semantik: Bausteine benötigen eine Bezeichnung und eine Beschreibung, die keinen Zweifel an ihrem Sinn und Funktion lassen

LZ 2-2: Unterschiedliche Arten von Bausteinen beschreiben und begründen

- Arten von Modulen: Definition und Eigenschaften verschiedener Bausteine, wie Microservices, Self-contained Systems und Deployment-Monolithen.
- Lebenszyklus von Modulen: Beschreibung, wie ein Baustein erstellt, integriert, getestet, deployt und skaliert wird.
- Isolationsebenen: Unterschiede zwischen modularer Isolation im Code, in der Laufzeitumgebung und bei Netzwerkschnittstellen.

LZ 2-3: Modularisierungskonzepte bewerten und auswählen

- Technische Modularisierung: Bewertung von Konzepten wie Dateien, Bibliotheken, Prozesse, Microservices oder Self-contained Systems.
- Integration und Kopplung: Analyse von Kopplungsebenen (Sourcecode, Kompilate, Netzwerkprotokoll) und deren Auswirkungen.
- Qualitätsziele berücksichtigen: Auswahl von Modularisierungskonzepten anhand von Anforderungen wie "Parallelisierbarkeit der Entwicklung" oder "unabhängiges Deployment von Modulen".

LZ 2-4: Modularisierungsstrategien bewerten

- Strategien und Konsequenzen: Bewertung der Auswirkungen verschiedener Modularisierungsansätze, z. B. Monolith vs. verteilte Module.

- Technologieabhängigkeit: Wie Modularisierungstechnologien (z. B. Container oder Laufzeitumgebungen) Strategien beeinflussen.
- Aufwands-Nutzen-Abgleich: Abwägung organisatorischer und technischer Aufwände gegen die erwarteten Vorteile.

LZ 2-5: Aufwand und Nutzen von Modularisierungsstrategien gegenüberstellen

- Integration vs. Dezentralisierung: Identifikation von organisatorischen und technischen Aufwänden für die Integration modularer Systeme.
- Erwartete Vorteile: Bewertung des Nutzens, wie etwa unabhängiges Deployment, Parallelisierung der Arbeit, Austauschbarkeit von Technologien und verbesserte Verständlichkeit durch klar benannte Module und Integrationspunkte.
- Modulgrenzen und Komplexität: Analyse, wie die Wahl der Modulgrenzen die Komplexität, den Abstimmungsaufwand bei Änderungen sowie die Entwicklungs- und Wartungskosten beeinflusst.

.

2.3. Referenzen

[Eric Evans 2003], [Lewis, Fowler, et al. 2013], [Wolff 2018], [Sam Newman 2021], [Parnas 1972]

3. Softwaremodule und der Bezug zu Organisationsstrukturen

Dauer: 90 Min.	Übungszeit: 60 Min.

3.1. Begriffe und Konzepte

Conway's Law, Kommunikationsstrukturen, Context Mapping, Strategic Design, Stakeholder Management, Team Topologies, Sozio-technische Architektur, Teamautonomie, Team- und Systemgrenzen, Makro- und Mikroarchitektur-Entscheidungen.

3.2. Lernziele

LZ 3-1: Wechselwirkung von Architekturtypen und Organisation analysieren und benennen

- Erklären, wie Beziehungen zwischen Teams bzw. Organisationseinheiten die Architektur von Systemen beeinflussen und damit Auswirkungen auf Abstimmungsaufwände und Time-to-Market haben
- Erkennen, wie Conway's Law die Entwicklungsgeschwindigkeit und Änderungsfähigkeit von Systemen beeinflusst
- Den Zusammenhang zwischen Teamgrenzen und Systemgrenzen analysieren und erklären
- [OPTIONAL] Die Balance zwischen technischer und organisatorischer Struktur erklären
- Unterschiede von Buildtime- und Runtime-Abhängigkeiten für Softwareentwicklungsprozesse erklären
- Die Parallelisierbarkeit von Softwareentwicklungsaufgaben als Architekturziel verstehen
- [OPTIONAL] Darstellen wie organisatorische Änderungen die Software beeinflussen und umgekehrt, um bewusste Entscheidungen für Organisations- und Softwarearchitektur treffen zu können.

LZ 3-2: Kommunikationsstruktur der Organisation bei Zerlegung berücksichtigen

- Conway's Law: Bedeutung verstehen und Auswirkungen der Kommunikationsstruktur der Organisation auf die Wahl und Gestaltung von Modulgrenzen.
- Autonomie und Zusammenarbeit: Sicherstellung der Entwicklungsautonomie durch modulare Schnitte entlang fachlicher Grenzen.

LZ 3-3: Context Maps für Stakeholder Management nutzen

- Context Maps aus Domain-Driven Design (DDD) einsetzen, um den aktuellen IST-Zustand und einen gewünschten zukünftigen SOLL-Zustand einer Systemlandschaft darzustellen.
- [OPTIONAL] Context Maps zielgruppenabhängig (Entwickler vs. Management) als Kommunikations- und Planungswerkzeug verstehen und anwenden.

LZ 3-4: [OPTIONAL] Begriffe wie Teamorganisation und sozio-technische Architekturen sicher verwenden

- Begriffe wie Team-Organisation, sozio-technische Architekturen und ähnliche Konzepte einordnen.
- Bedeutung im Kontext moderner Softwareentwicklung erklären.
- Verständnis für die Verknüpfung technischer und sozialer Aspekte bei der Architekturgestaltung zu entwickeln.

LZ 3-5: Außerhalb des eigenen Einflussbereiches getroffene Makroarchitekturentscheidungen identifizieren

- Unterschied zwischen Mikro- und Makroarchitektur verstehen, um potenzielle Einschränkungen und Abstimmungsaufwände zu identifizieren und proaktiv zu adressieren.
- Die Abgrenzung zwischen Makro- und Mikroarchitektur verstehen, um potenzielle Einschränkungen und Möglichkeiten bei Deployment-Strategien zu identifizieren.
- Makroarchitekturentscheidungen wie Kommunikationsprotokolle, Betriebsstandards und Plattformvorgaben analysieren und deren Auswirkungen auf Deployment- und Runtime-Methoden bewerten.
- Die Bedeutung von zentralen Plattformstandards für die Effizienz von Kommunikation und Deployments/Softwarelieferungen erkennen.

3.3. Referenzen

[Skelton, Pais 2019], [Eric Evans 2003], [Vossen, Haselmann, Hoeren 2012], [Conway 1968], [Baxter, Sommerville 2011]

4. Integrationsmethoden & Protokolle

Dauer: 150 Min.	Übungszeit: 60 Min.

4.1. Begriffe und Konzepte

Strategic Design (DDD), Kollaborationsmuster (z. B. Anti-Corruption Layer, Shared Kernel, Customer/Supplier, …), Frontend Integration, Middle-Tier Integration (REST, RPC, Messaging), Datenbankintegration, Legacy Systeme, CAP Theorem, ACID, BASE, Authentifizierung, Autorisierung, (lose) Kopplung, Skalierbarkeit, Messaging Patterns (Request/Reply, Publish/Subscribe), Domain Events, dezentrale Datenhaltung.

4.2. Lernziele

LZ 4-1: Integrationsstrategien gegenüberstellen (am Beispiel von DDD Strategic Design)

- Strategic Design aus dem Domain-Driven Design (DDD) und dessen Beziehungspatterns (z. B. Anti-Corruption Layer, Shared Kernel, Customer/Supplier) verstehen und erklären.
- Modulgrenzen eines Systems anhand von "Bounded Context" darstellen, beschreiben und SOLL/IST-Vergleiche anstellen.
- Anhand der Qualitätsziele begründen welche Patterns aus dem Strategic Design für eine Integration/Schnittstelle verwendet werden können und welche Patterns nicht.

LZ 4-2: Technische Integrationsmechanismen auswählen und begründen

- Bewertung der Vor- und Nachteile von technischen Integrationsmechanismen wie Frontend-Integration, Middle-Tier-Integration (REST, RPC, Messaging) oder Datenbankintegration anhand spezifischer Qualitätsziele des System und notwendiger Erfahrung/Skills der betreuenden Entwicklungsteams.
- [OPTIONAL] Frontend-Integration (Links, Client-Side Includes, Micro-Frontends, etc.) und Middleware für Legacy-Systeme sowie deren Auswirkungen vergleichen.
- [OPTIONAL] Praktische Anwendung in brow-field Szenarien: Integration von Legacy-Datenmodellen durch Transformation und Mapping.

LZ 4-3: Konsistenzmodelle erklären und auswählen (CAP Theorem)

- Erklärung des CAP-Theorems: Abwägung zwischen Konsistenz, Verfügbarkeit und Partitionstoleranz.
- Notwendigkeit von Partitionstoleranz verstehen.
- Vergleich von ACID- und BASE-Modellen: Garantien, Verluste und Tradeoffs sowie daraus resultierende Einsatzmöglichkeiten in verteilten Systemen verstehen.
- Unterschied zwischen "Consistency" bei ACID und BASE anhand praktischer Beispiele erklären.
- [OPTIONAL] Praxisbeispiel zu ACID-Transaktionen und BASE-Transaktionen in verteilten Systemen.
- CP- vs. AP-Systemdesign anhand von Qualitätszielen begründen.
- [OPTIONAL] Praxisbeispiel: Wahl eines geeigneten Konsistenzmodells basierend auf Systemanforderungen wie Latenz und Skalierbarkeit.
- [OPTIONAL] Produktbeispiele aus unterschiedlichen Kategorien (z. B. NoSQL, Konfigurationswerkzeuge, Service Discovery) kennen.

LZ 4-4: Resilience Patterns benennen

- [OPTIONAL] Messaging Patterns: Request/Reply, Publish/Subscribe und deren Resilience-Eigenschaften.
- Resilience-Strategien für dezentrale Datenhaltung und redundante Architektur.
- Unterschiede zwischen High-Availability und Resilience verstehen und Praxisbeispiele nennen
- [OPTIONAL] Nutzung von Mechanismen wie Circuit Breaker, Bulkhead und Graceful Degradation zur Sicherstellung der Verfügbarkeit in Szenarien, wo Systemteile unterschiedliche Verfügbarkeitsanforderungen haben.

LZ 4-5: Sicherheitsauswirkungen von Integrationsmethoden kennen und berücksichtigen

- [OPTIONAL] Vergleich von Authentifizierungs- und Autorisierungsmechanismen (z. B. OAuth, Kerberos).
- Auswirkungen von synchroner (z. B. RPC) vs. asynchroner Integration (z. B. Messaging) auf Sicherheit und Datenintegrität.
- Implementierung sicherer Schnittstellen und Makroarchitektur für verteilte Systeme.

LZ 4-6: [OPTIONAL] Event-getriebene Architekturen kennen und entwerfen

- Einführung in Event-Driven Architectures (EDA): Publizieren von Domain Events zur Entkopplung von Systemen.
- Entwurf einer EDA mit Messaging-Systemen wie RabbitMQ oder Kafka.
- Herausforderungen und Lösungen bei Backpressure und dezentraler Datenhaltung.

4.3. Referenzen

[Eric Evans 2003], [Parecki et al. 2006], [Hohpe, Woolf 2003], [Tanenbaum, van Steen 2006], [Lamport 1998], [Brewer 2000], [Takada 2013], [Nygard 2018], [Hanmer 2007], [Hamilton 2007]

5. Installation und Laufzeitumgebungen/Plattformen

Dauer: 90 Min.	Übungszeit: 90 Min.

5.1. Begriffe und Konzepte

Continuous Deployment, DevOps, Platform Engineering, Infrastructure as Code, Konfigurationsmanagement, Kosteneffizienz, Zero-Downtime Deployments.

5.2. Lernziele

LZ 5-1: Voraussetzungen und Auswirkungen für Continuous Deployment benennen

- Anforderungen an eine CI/CD-Pipeline erklären, einschließlich Automatisierungsgrad, Tests und Infrastrukturintegration.
- Die Rolle von DevOps in Continuous Deployment verstehen und organisatorische Implikationen erkennen.
- Unterschiede zwischen traditionellen und modernen Deployment-Ansätzen, wie Immutable Infrastructure oder Infrastructure as Code, evaluieren.
- Risiken und Vorteile von Continuous Deployment, einschließlich Zero Downtime, im Kontext verschiedener Projekte abwägen.

LZ 5-2: [OPTIONAL] Unterschiede von IaaS, PaaS, CaaS, FaaS erklären und auswählen

- Die technischen Eigenschaften und Anwendungsbereiche der verschiedenen Plattformansätze (IaaS, PaaS, CaaS, FaaS) erläutern.
- Kriterien für die Auswahl des passenden Plattformansatzes in Abhängigkeit von Projektanforderungen wie Skalierbarkeit, Flexibilität und Kosten aufstellen.
- Auswirkungen der Plattformwahl auf die Deployment-Strategie analysieren und Entscheidungen zu Containerisierung (z. B. Docker, Kubernetes) oder serverlosen Ansätzen treffen.

LZ 5-3: Zero Downtime Methodiken und ihre Auswirkungen benennen und auswählen

- Verschiedene Zero-Downtime-Strategien (Blue-Green-Deployment, Canary Releases, Rolling Updates) erläutern und deren Vor- und Nachteile analysieren.
- Die Rolle von Immutable Infrastructure und Automatisierung für Zero Downtime erklären.
- Herausforderungen und Architekturanforderungen für Zero-Downtime-Deployments evaluieren.

LZ 5-4: Unterschiede zwischen Continuous Integration, Continuous Deployment und Continuous Delivery erklären

- Die Bedeutung von Continuous Integration (CI) als Grundlage für Continuous Deployment und Delivery erläutern.
- Unterschiede in Zielsetzung und Automatisierungsgrad zwischen den drei Konzepten analysieren.
- Abhängigkeiten von Architekturentscheidungen auf den Einsatz von CI/CD-Pipelines beschreiben.

LZ 5-5: [OPTIONAL] Deployment-spezifische Sicherheitsanforderungen benennen

- Anforderungen an den Schutz von Secrets (z. B. API-Keys, Zertifikate) und Sicherheitsrichtlinien im Deployment-Prozess erklären.

- Konzepte wie Zugriffskontrollen, Verschlüsselung und Compliance-Anforderungen in Deployment-Strategien integrieren.
- Werkzeuge zur Verwaltung sicherheitsrelevanter Informationen (Secret Store) einsetzen und bewerten.

LZ 5-6: [OPTIONAL] Die Rolle von Observability im Deployment-Prozess erklären

- Anforderungen an ein Monitoring-System und dessen Bedeutung für den Deployment-Erfolg erläutern.
- Die Rolle von zentralisierten Logging- und Metriksystemen (z. B. Elastic Stack, Prometheus, Grafana) erklären.
- Strategien entwickeln, um potenzielle Probleme im Deployment-Prozess frühzeitig zu erkennen und zu adressieren.

LZ 5-7: [OPTIONAL] Kosten- und Ressourceneffizienz im Deployment optimieren

- Methoden zur Kostenoptimierung im Deployment evaluieren (z. B. Reserved Instances, Spot Instances, Serverless).
- Auswirkungen der Wahl von Cloud- und Virtualisierungsstrategien auf die Betriebskosten und Ressourcennutzung analysieren.
- Deployment-Methoden entwickeln, die Skalierbarkeit und Effizienz maximieren.
- Negative Auswirkungen von Kostenoptimierung auf Flexibilität, Reaktionsfähigkeit und Geschwindigkeit im Deployment Prozess erklären

5.3. Referenzen

[Vossen, Haselmann, Hoeren 2012], [Wolff, Müller, Löwenstein 2013], [Humble, et al. 2010], [Wolff 2016]

6. Betriebsmodelle

Dauer: 90 Min.	Übungszeit: 60 Min.

6.1. Begriffe und Konzepte

Observability, Monitoring, Betriebsmodelle, MTBF vs. MTTR, Logging, Tracing, Metrics, Alerting, Service Level Objectives, Chaos Engineering.

6.2. Lernziele

LZ 6-1: Unterschiedliche Betriebsmodelle und ihre Auswirkungen erklären und auswählen

- Operations Team vs. You Build It, You Run It:
  - Vergleich zentralisierter Betriebsmodelle (dediziertes Operations-Team) mit dezentralisierten Ansätzen wie DevOps und "You Build It, You Run It".
  - Auswirkungen der Betriebsmodelle auf Verantwortlichkeiten, Kommunikationswege und Problemlösungszeiten.
  - Die Rolle von DevOps-Praktiken bei der Integration von Betrieb und Entwicklung.
  - Anhand des Kontextes - wie Organisation und Skillset - für einen zentralen oder dezentralen Anwendungsbetrieb entscheiden.
- MTBF (Mean Time Between Failures) vs. MTTR (Mean Time to Repair):
  - Bedeutung der beiden Metriken und deren Auswirkung auf Betriebsmodelle und -kultur.
  - [OPTIONAL] Strategien zur Optimierung von MTBF und MTTR im Kontext der gewählten Architektur und ihre Auswirkungen auf Continuous Deployment.
  - Zusammenhang zwischen Betriebsmodellen und Qualitätskriterien wie Änderungsfähigkeit, Zuverlässigkeit und Fehlertoleranz.

LZ 6-2: Observability – Unterschiede von Metriken, Logs und Traces verstehen und richtig einsetzen

- Definition und Bedeutung von Observability:
  - Abgrenzung zu Monitoring und Erklärung der Schlüsselaspekte von Observability.
  - Die Rolle von Logs, Metriken und Traces bei der Fehlersuche und Systemüberwachung/-verständnis.
  - Die Teilnehmer sollten die Unterschiede zwischen Logging, Monitoring und Metrik-Sammlung verstehen sowie die damit einhergehenden Unterschiede bei den für diese Aufgaben verwendeten Werkzeugen.
- Einsatz von Logs, Metriken und Traces:
  - Beispiele für typische Anwendungsfälle
  - Werkzeuge und Technologien für Observability (z. B. Elastic Stack, Prometheus, Jaeger).
  - Logs sind Events, Metriken sind Zustände zu einem bestimmten Zeitpunkt.
  - Monitoring kann sowohl fachliche als auch technische Daten enthalten.
  - Die richtige Auswahl von Daten ist zentral für ein zuverlässiges Verständnis des Runtime-Verhaltens des gebauten Systems.

- Die Teilnehmer sollen verstehen, welche Informationen sie aus Traces, welche aus Log-Daten und welche sie (besser) durch Instrumentierung des Codes mit Metrik-Sonden beziehen können.

- Architekturelle Unterstützung für Observability:
  - Architekturentscheidungen, die die Integration und Nutzung von Observability-Tools erleichtern.
  - Unterstützung des Betriebs als Bestandteil der Architekturkonzepte
  - zentralistische Logdaten-Verwaltung und Alternativen sowie deren Auswirkungen auf die Architektur (Performance-Overhead, Speicherverbrauch, Latenzen, Locking vs. Log-Verlust).
  - Aufbau von typischen Metrik-Architekturen (Erfassen, Sammeln & Samplen, Persistieren, Abfragen, Visualisieren)
  - Unterscheidung zwischen Geschäfts-, Anwendungs- und Systemmetriken.
  - Bedeutung wichtiger, werkzeugunabhängiger System- und Anwendungsmetriken.

LZ 6-3: Fehleranalyse in verteilten Systemen erleichtern

- Herausforderungen in verteilten Systemen:
  - Typische Probleme wie Netzwerklatenzen, Partial Failures und inkonsistente Zustände.
  - Komplexität der Fehleranalyse durch asynchrone Kommunikation und dezentrale Datenhaltung.
- Werkzeuge und Strategien zur Fehleranalyse:
  - Einsatz von verteiltem Tracing (z. B. OpenTelemetry, Jaeger) zur Visualisierung von Request-Flows.
  - Logging- und Monitoring-Strategien für dezentrale Systeme.
  - Bedeutung zentralisierter Datenplattformen (z. B. Log-Aggregation) für die Fehlerdiagnose.
  - Die Teilnehmer sollen Architekturvorgaben so treffen können, dass der Einsatz geeigneter Werkzeuge bestmöglich unterstützt, dabei jedoch angemessen mit Systemressourcen umgegangen wird.
  - Dabei können sie abhängig vom konkreten Projekt-Szenario den Fokus im Konzept auf Logging, Monitoring und die dazu notwendigen Daten legen.
- Architekturmuster zur Unterstützung der Fehleranalyse:
  - Patterns wie Circuit Breaker, Retry-Mechanismen und Bulkhead zur Fehlerisolation.
  - Architekturentscheidungen für transparente Fehlermeldung und Fehlerweitergabe.

LZ 6-4: [OPTIONAL] Service Level Objectives (SLOs) aus Qualitätszielen ableiten

- Definition und Bedeutung von SLOs:
  - Unterschied zwischen SLOs, SLAs (Service Level Agreements) und SLIs (Service Level Indicators).
  - Ableitung von SLOs aus Qualitätszenarien, nichtfunktionalen Anforderungen und Geschäftsanforderungen.
- Architekturelle Unterstützung für SLOs:
  - Einfluss von Architekturentscheidungen auf die Einhaltung von SLOs (z. B. Skalierbarkeit, Redundanz).
  - Beispiele für typische SLOs wie Verfügbarkeit, Antwortzeit und Fehlerrate.

- Werkzeuge zur Überwachung und Messung von SLOs:
  - Automatisierte Überwachungs- und Berichtsmechanismen für SLO-Tracking.

LZ 6-5: [OPTIONAL] Mit Architektur das Incident Management und schnelle MTTR unterstützen

- Architekturentscheidungen zur Reduzierung der MTTR:
  - Einsatz von Self-Healing-Mechanismen und automatisierter Wiederherstellung.
  - Unterstützung durch observability-fokussierte Architekturentscheidungen.
- Incident Management und Resilienz:
  - Architekturmuster wie Circuit Breaker, Fallback-Strategien und Rate Limiting zur Minimierung von Ausfällen.
  - Einfluss von Deployments und Rollbacks auf die Reaktionszeiten bei Incidents.

LZ 6-6: [OPTIONAL] Durch Architektur zu Disaster Recovery und Business Continuity Management beitragen

- Disaster Recovery Strategien:
  - Einsatz von Backup- und Wiederherstellungsmechanismen (Snapshots, Datenbank-Replikation).
  - Anforderungen an Architekturen für Recovery-Ziele, wie RTO (Recovery Time Objective) und RPO (Recovery Point Objective).
- Business Continuity Management (BCM) für Architekten:
  - Architekturelle Unterstützung für hohe Verfügbarkeit und Ausfallsicherheit.
  - Technologien wie Multi-Region-Deployments, Failover-Strategien und Geo-Redundanz.
- Testen von Disaster-Recovery-Strategien:
  - Implementierung und Validierung von Recovery-Plänen durch Simulationen und Probeläufe.

LZ 6-7: [OPTIONAL] Chaos Engineering anhand von Qualitätszielen designen und durchführen

- Grundlagen des Chaos Engineering: Zielsetzung und Prinzipien (z. B. "Testen in Produktion" zur Erhöhung der Resilienz).
- Design von Chaos-Experimenten:
  - Ableitung von Experimenten aus Qualitätszielen wie Verfügbarkeit und Fehlertoleranz.
  - Identifikation von Schwachstellen und deren Validierung durch gezielte Störungen.
- Architekturunterstützung für Chaos Engineering:
  - Einsatz von resilienzfördernden Mustern wie Circuit Breaker und Fallback-Mechanismen.
  - Integration von Chaos-Testing-Werkzeugen (z. B. Gremlin, Chaos Monkey) in die CI/CD-Pipeline.

6.3. Referenzen

[Wolff 2016], [Nygard 2018]

Referenzen

Dieser Abschnitt enthält Quellenangaben, die ganz oder teilweise im Curriculum referenziert werden.

B

- [Baxter, Sommerville 2011] Gordon Baxter, Ian Sommerville: Sociotechnical Systems Design: Evolving Theory and Practice, 2011, https://academic.oup.com/iwc/article/23/1/4/693091
- [Brewer 2000] Eric Brewer, Towards Robust Distributed Systems, PODC Keynote, July-19-2000

C

- [Conway 1968] Melvin E. Conway, How Do Committees Invent?, Datamation, 1968, https://en.wikipedia.org/wiki/Conway%27s_law

E

- [Eric Evans 2003] Eric Evans: Domain-Driven Design: Tackling Complexity in the Heart of Software, Addison- Wesley Professional, 2003

H

- [Hamilton 2007] James Hamilton, On Designing and Deploying Internet-Scale Services, 21st LISA Conference 2007
- [Hanmer 2007] Robert S. Hanmer, Patterns for Fault Tolerant Software, Wiley, 2007
- [Hohpe, Woolf 2003] Gregor Hohpe, Bobby Woolf: Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions, Addison-Wesley, 2003, ISBN 978-0-32120-068-6
- [Humble, et al. 2010] Jez Humble, David Farley: Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation, Addison-Wesley, 2010, ISBN 978-0-32160-191-9
- [Humble, et al. 2014] Jez Humble, Barry O'Reilly, Joanne Molesky: Lean Enterprise: Adopting Continuous Delivery, DevOps, and Lean Startup at Scale, O'Reilly 2014, ISBN 978-1-44936-842-5

L

- [Lewis, Fowler, et al. 2013] James Lewis, Martin Fowler, et al.: Microservices - http://martinfowler.com/articles/microservices.html, 2013
- [Lamport 1998] Leslie Lamport, The Part-Time Parliament, ACM Transactions on Computer Systems 16, 2 (May 1998), 133-169

N

- [Sam Newman 2021] Sam Newman: Building Microservices: Designing Fine-Grained Systems, O'Reilly Media, 2nd ed. edition, 2021, ISBN 978-1-49203-402-5
- [Nygard 2018] Michael T. Nygard, Release It!, 2. Auflage, Pragmatic Bookshelf, 2018

O

- [Parecki et al. 2006] Aaron Parecki et al., Explaining OAuth 2.0 http://oauth.net/

P

- [Parnas 1972] David Parnas, On the Criteria To Be Used in Decomposing Systems into Modules. Communications of the ACM 15(12):1053—1058, 1972.

T

- [Takada 2013] Mikito Takada, Distributed Systems for Fun and Profit, http://book.mixu.net/distsys/ (Guter Einstieg und Überblick)
- [Tanenbaum, van Steen 2006] Andrew Tanenbaum, Marten van Steen, Distributed Systems – Principles and Paradigms, Prentice Hall, 2nd Edition, 2006
- [Skelton, Pais 2019] Mathew Skelton, Manuel Pais - TEAM TOPOLOGIES: ORGANIZING BUSINESS AND TECHNOLOGY TEAMS FOR FAST FLOW, IT Revolution, 2019, ISBN 978-1-942788-81-2

V

- [Vossen, Haselmann, Hoeren 2012] Gottfried Vossen, Till Haselmann, Thomas Hoeren: Cloud-Computing für Unternehmen: Technische, wirtschaftliche, rechtliche und organisatorische Aspekte, dpunkt, 2012, ISBN 978-3-89864-808-0

W

- [Wolff 2016] Eberhard Wolff: Continuous Delivery: Der pragmatische Einstieg, 2. Auflage, dpunkt, 2016, ISBN 978-3-86490-371-7
- [Wolff 2018] Eberhard Wolff: Microservices - Grundlagen flexibler Software Architekturen, 2. Auflage, dpunkt, 2018, ISBN 978-3-86490-555-1
- [Wolff, Müller, Löwenstein 2013] Eberhard Wolff, Stephan Müller, Bernhard Löwenstein: PaaS - Die wichtigsten Java Clouds auf einen Blick, entwickler.press, 2013