Curriculum für

Certified Professional for Software Architecture (CPSA)®
Advanced Level

Modul
CLOUDINFRA

Infrastruktur, Container und Cloud Native

2024.1-rev0-DE-20240516

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

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 Expert:innen 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 „CLOUDINFRA“?

Microservices, Container und Container-Manager haben die Art und Weise wie wir Software konzipieren, entwickeln und in Produktion bringen in den letzten Jahren stark verändert. Moderne Applikationen müssen in einem Cluster von mehreren Knoten funktionieren, dynamisch platzierbar, skalierbar und fehlertolerant sein.

Die Teilnehmenden lernen in diesem Modul Wege zur Realisierung dynamischer Cloud-Native-Architekturen, Container Application Design, Logging/Monitoring/Alerting, Container Native Storage und Möglichkeiten zur UI-Integration. Ebenso werden typische Konzepte aktueller Container-Manager aufgezeigt und vermittelt, wie sich damit für größere Webanwendungen gängige Qualitätsanforderungen realisieren lassen. Zusätzlich werden gängige Services verschiedener Cloud-Anbieter besprochen, Möglichkeiten zur Automatisierung aufgezeigt, Ansätze der Softwareentwicklung und des Application Lifecycle besprochen.

Bei CLOUDINFRA stehen betriebliche Aspekte im Vordergrund. Die im Modul FLEX detaillierten Konzepte zu Entwurf und Umsetzung von Architekturen werden, wo zum Verständnis notwendig ist, übersichtsweise erläutert.

Struktur des Lehrplans und empfohlene zeitliche Aufteilung

Inhalt	Empfohlene Mindestdauer (min)
1. Grundlagen moderner Infrastrukturen	180
2. Gängige Architekturkonzepte	120
3. Cloud Native Journey	240
4. Muster für verteilte Anwendungen und Cloud Native Architekturen	240
5. Development und CI/CD	150
6. Automatisierung und Betrieb	150
7. Case Study	120
Summe	1200 (20h)

Dauer, Didaktik und weitere Details

Die unten genannten Zeiten sind Empfehlungen. Die Dauer einer Schulung zum Modul CLOUDINFRA 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 CLOUDINFRA 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

Teilnehmende **sollten** folgende Kenntnisse und/oder Erfahrung mitbringen:

- Praktische Erfahrung in Entwurf und Entwicklung kleiner bis mittelgroßer Softwaresysteme
- Erste praktische Erfahrung in Wartung oder Weiterentwicklung von Softwaresystemen
- Erste praktische Erfahrung im Umgang mit Containern und deren Deployment

Hilfreich für das Verständnis einiger Konzepte sind darüber hinaus:

- Kenntnisse oder erste praktische Erfahrung in der Herleitung und Inbetriebnahme moderner Microservice-Architekturen
- Erste praktische Erfahrung im Umgang mit Container-Managern
- Erste praktische Erfahrung mit gängigen Cloud-Anbietern

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.

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. Grundlagen moderner Infrastrukturen

Dauer: 180 Min.	Übungszeit: 0 Min.

1.1. Begriffe und Konzepte

Cloud, Cloud-Arten, Cloud-Anbieter, On Premise, Bare Metal, Cloud Service Modelle (*aaS), Vendor Lock-in, Managed Services, Cloud Native Services, Cloud-Muster, Cloud-Migrationsmuster, Hybrid/Multi Cloud, Organisatorische Aspekte der Cloud Migration, Rechtliche Rahmenbedingungen, Time-to-Market, Verfügbarkeit, Skalierung, Geo Redundanz, Performance, IOPS, Decoupling Operations, Networking.

1.2. Lernziele

LZ 1-1: Cloud-Arten verstehen und unterscheiden

Softwarearchitekt:innen kennen die gängigen Cloud-Begriffe und können verschiedene Arten wie Public, Private, Hybrid/Multi und On-Premise voneinander unterscheiden.

Sie verstehen, dass der Hybrid/Multi Cloud Betrieb keine Ausnahme darstellt und können die Gründe von Hybrid/Multi Cloud Betrieb benennen, beispielsweise:

- Schrittweise Migration in die Cloud
- Kostenersparnis beim Betrieb
- Datenschutz und Datensicherheit
- Integration mit Bestandssystemen

Sie verstehen den Unterschied zwischen Public Cloud- und On-Premise-Betrieb sowie die Herausforderungen und Argumente für und gegen den Public Cloud-Betrieb. Sie erkennen auch, wann der Einsatz von eigener Hardware (Bare Metal) sinnvoller ist.

LZ 1-2: Entscheidungskriterien für die Nutzung von managed Services und self-managed Services verstehen

Softwarearchitekt:innen kennen unterschiedliche Cloud Service Modelle (*aaS) und können Services anhand dieser Modelle klassifizieren.

Sie verstehen darüber hinaus das Serverless Computing Konzept und können es den Cloud Service Modellen zuordnen.

Softwarearchitekt:innen verstehen das Shared-Responsibility-Modell und dessen Relevanz für Kosten- und Risikobewertungen bei der Nutzung von managed Cloud Services.

Sie kennen das Konzept des Vendor Lock-in und seine Relevanz für die Entscheidungsfindung zwischen managed und self-managed Services.

LZ 1-3: Compliance und organisatorische Aspekte bei der Cloud-Architektur berücksichtigen

Softwarearchitekt:innen kennen die organisatorischen Aspekte der Cloud Migration und die rechtlichen Rahmenbedingungen für einen Betrieb von Anwendungen in der Cloud, z. b.:

- DSGVO (GDPR)
- BSI C5 (Kriterienkatalog Cloud Computing)
- ISO 27001
- BSI IT-Grundschutz

Sie verstehen, dass die Aspekte der Datensicherheit für die Entscheidung für oder gegen eine Public Cloud relevant sind und dass organisatorische und/oder rechtliche Anforderungen die Wahl der verwendeten Cloud-Art beeinflussen.

LZ 1-4: Verfügbarkeit, Skalierung und Performance als Anforderungsparameter beim Cloud-Architektur-Design anwenden

Softwarearchitekt:innen kennen die Auswirkungen der Cloud auf Time-to-Market, Verfügbarkeit, Skalierung, Performance, IOPS, Decoupling Operations und Networking.

Sie können einschätzen, wann Cloud-Anwendungen eine Verteilung über mehrere Verfügbarkeitszonen und Rechenzentren erfordern.

Sie kennen die Leistungsanforderungen an die Infrastruktur und die Limitierungen der Public Cloud, wie z.B. IOPS für Speicher.

2. Gängige Architekturkonzepte

Dauer: 120 Min.	Übungszeit: 20 Min.

2.1. Begriffe und Konzepte

Self-contained Systems, Microservices, Independent Systems Architecture, Integrationskonzepte, Service Discovery, CQRS, Event Sourcing, Eventual Consistency.

2.2. Lernziele

LZ 2-1: Situativ passende Kommunikations-Konzepte verteilter Anwendungen auswählen

Softwarearchitekt:innen kennen Muster moderner Architekturen für verteilte Anwendungen (speziell Microservices, Self-contained Systems) und verstehen die Gründe, die zu diesen Lösungen führen.

- Monolith
- Microservices
- Self-contained Systems

Sie verstehen die unterschiedlichen Technologien entkoppelter Kommunikation und des Datenaustauschs zwischen Services, zum Beispiel:

- Event Sourcing
- Messaging Middleware
- HTTP feeds

Softwarearchitekt:innen kennen die Auswirkungen einer Shared-Something/Nothing-Architektur, die insbesondere durch eine Skalierung über Container entsteht.

LZ 2-2: Situativ passende Integrationskonzepte verteilter Anwendungen auswählen

Softwarearchitekt:innen verstehen, wie mehrere Services über unterschiedliche Wege zu einer Applikation integriert werden können, zum Beispiel durch Integration über:

- Backend (message bus, database, etc.)
- Frontend (UI integration)

Softwarearchitekt:innen kennen die Auswirkungen von Eventual Consistency auf die Datenintegrität und Datenkonsistenz.

2.3. Referenzen

[Dehghani, Z.: How to break a monolith into microservices. martinfowler.com],[Cornelia Davis: Cloud Native Patterns], [Awati, R. & Wigmore, I.: Monolithic architecture. WhatIs.com], [Monolithic architecture pattern. microservices.io], [Fowler, M.: MonolithFirst. martinfowler.com], [Lewis, J.: Microservices. martinfowler.com], [Microservice Architecture Pattern. microservices.io], [SCS: Self-contained systems], [Wolff, E.: Self Contained Systems (SCS)], [Event sourcing. microservices.io], [Cloud Native landscape], [Messaging. microservices.io], [RSS 2.0 specification (Current)], [Backend integration. microservice-api-

patterns.org], [Frontend Integration. microservice-api-patterns.org]

3. Cloud Native Journey

Dauer: 240 Min.	Übungszeit: 30 Min.

3.1. Begriffe und Konzepte

Qualitätsanforderungen an die Plattform, Cloud Native Storage, Backup & Restore, Overlay Networking, Network Policies, Container Security, Container Linux, Cloud Migration.

3.2. Lernziele

LZ 3-1: Nutzen und Ziele des Cloud Native Konzepts verstehen

Softwarearchitekt:innen verstehen den Nutzen und die Ziele des Cloud Native Konzepts. Sie kennen die Vorteile des Konzepts insbesondere durch:

- Einheitliche Schnittstellen zur Administration und Konfiguration
- Ressourcen-Abstraktion
- Lieferartefakte
- Isolation der Applikation vom Hostsystem

LZ 3-2: Grundlegende Container Konzepte verstehen

Softwarearchitekt:innen verstehen die technischen Grundlagen und den Mehrwert von Containern und kennen den Unterschied zwischen konventionellem Deployment und dem mit Containern.

Sie verstehen, wie die Verwendung von Containern zu einer besseren Entkopplung von Verantwortlichkeiten und damit zu einer effektiveren Organisation führt.

Softwarearchitekt:innen verstehen Linux-Kernel-Isolationsmechanismen, die bei Containern Anwendung finden (Container Security), u. a.:

- Namespaces und cgroups
- Capabilities

Sie kennen außerdem die Auswirkungen, Nutzungszenarien und Gefahren privilegierter Container.

LZ 3-3: Abstraktionskonzepte von Container-Managern verstehen

Softwarearchitekt:innen verstehen die Abstraktionskonzepte aktueller Container-Manager und kennen bewährte Muster und Praktiken, wie die Erfüllung von Qualitätsanforderungen auf sie übertragen werden kann. Zum Beispiel:

- Finite Workloads wie (Cron-) Jobs
- Health Check und Self Healing
- Skalierung und Load Balancing
- Placement

Softwarearchitekt:innen kennen die Prinzipien eines Overlay-Networks (VLAN) für Container und Container-Manager und verstehen, wie verschiedene Services oder Systeme isoliert werden. Sie kennen Anforderungszenarien wie z. B. die Realisierung einer verteilten Firewall zur Isolation von:

- Mehreren Mandanten
- Testsystemen im Rahmen von CI/CD

LZ 3-4: Vorteile spezialisierter Linux-Distributionen für den Betrieb von Containern verstehen

Softwarearchitekt:innen verstehen, wie aktuelle Core/Container-Linux-Distributionen auf den Betrieb von Containern optimiert sind, welche Vorteile sie bieten (u. a. bzgl. Kernel- und Distributionsupdates) und welche Folgen das für die betriebenen Applikationen hat.

LZ 3-5: Persistenzlösungen in Cloud-Umgebungen bewerten und auswählen

Softwarearchitekt:innen kennen verschiedene Wege, um Daten zu persistieren.

Sie kennen den Unterschied von Object- und Block-Storage und können verschiedene Storage-Dienste der Cloud-Anbieter, unterschiedliche COTS-Produkte und selbstverwaltete Lösungen für unterschiedliche Einsatzszenarien klassifizieren.

Softwarearchitekt:innen verstehen die Vor- und Nachteile einer Cloud-Native-Storage-Lösung und können die höherwertigen Managed Services (wie etwa RDBMS) der Cloud-Anbieter mit der Verwendung selbstverwalteter Lösungen vergleichen.

Sie kennen verschiedene Konzepte, um Backup und Restore von persistenten Daten in der Cloud zu realisieren.

LZ 3-6: Cloud Migrationsmuster verstehen

Softwarearchitekt:innen kennen verschiedene Cloud-Migrationsmuster und können sie Anforderungsgetrieben auswählen.

Sie kennen darüber hinaus Migrationsmuster, die ein Hybrid/Multi Cloud Setup erfordern, z. B. für eine schrittweise Migration in die Cloud.

3.3. Referenzen

[Cornelia Davis: Cloud Native Patterns], [Brendan Burns: Designing Distributed Systems], [Ibryam Bilgin, Roland Huss: Kubernetes Patterns.]

4. Muster für verteilte Anwendungen und Cloud Native Architekturen

Dauer: 240 Min.	Übungszeit: 20 Min.

4.1. Begriffe und Konzepte

Resilienz Muster, Container Application Design, Cloud Native Architekturen, Container Pattern, Functions as a Service (FaaS), Service Mesh

4.2. Lernziele

LZ 4-1: Verschiedene Muster für die Modularisierung von Cloud-Native Architekturen kennen

Softwarearchitekt:innen kennen die Konzepte des Container Application Designs (Container Pattern), um Softwarekomponenten in Container modularisieren zu können und Cloud-native Architekturen zu realisieren.

Sie kennen verschiedene Container Application Design Muster, wie zum Beispiel:

- Ambassador/Adapter/Sidecar
- Scatter & Gather
- Work Queue

Softwarearchitekt:innen kennen Methoden, um technische von fachlichen Aufgaben durch getrennte Container zu realisieren und verstehen, wie Container-Manager über dieses Prinzip technische Aufgaben übernehmen, insbesondere:

- Konfiguration und Initialisierung von Anwendungen
- Adaptive Skalierung über Custom Metrics
- Container Management durch Operator bzw. Controller

LZ 4-2: Passende Technologien für den Betrieb von Modulen auswählen

Softwarearchitekt:innen können geeignete Technologien zum Betrieb von Modulen eines verteilten Systems auswählen, z.B. mit Functions as a Service (FaaS) und Container Orchestrierung.

Darüber hinaus können Sie die Anwendung dieser Technologien Anforderungsgetrieben anwenden. Dabei gibt es verschiedene Aspekte zu beachten wie:

- Skalierungsanforderungen
- Start- und Ausführungszeit
- Komplexität des Betriebs und fachlichen Schnitts
- Zugriff auf Persistenz
- Limitierungen für Observability und Debugging

LZ 4-3: Passende Resilienzmuster zur Erhöhung von Fehlertoleranz auswählen

Softwarearchitekt:innen verstehen, wie bei einer verteilten Anwendung die Kommunikation zwischen den Services fehlertolerant realisiert werden kann.

Sie verstehen, durch welche Muster Fehlertoleranz auf der Kommunikationsebene erhöht werden kann und wie gängige Service-Mesh-Konzepte verwendet werden können, um die Implementation von Resilienzmustern vom fachlichen Code zu trennen.

Softwarearchitekt:innen kennen u. a. folgende Resilienzmuster:

- Circuit Breaker
- Conditional Rate Limits
- Traffic Shifting

Sie kennen das Konzept von Chaos Engineering und dessen Methoden, die Systemresilienz durch gezieltes Einbringen von Störungen zu testen.

4.3. Referenzen

[Cornelia Davis: Cloud Native Patterns], [Brendan Burns: Designing Distributed Systems], [Ibryam Bilgin, Roland Huss: Kubernetes Patterns.]

5. Development und CI/CD

Dauer: 150 Min.	Übungszeit: 20 Min.

5.1. Begriffe und Konzepte

Development Environment, CI/CD Environment, Mean Time To Recovery (MTTR), Application Lifecycle Management, Formen von Deployments wie Rolling-, Canary- und Blue/Green Deployment, Cluster Design, Secrets Management

5.2. Lernziele

LZ 5-1: Projekte in Cloud-Umgebungen umsetzen

Softwarearchitekt:innen wissen, dass es bei der Arbeit in Cloud-Umgebungen neue Anforderungen an den Softwareentwicklungsprozess gibt.

Sie kennen verschiedene Vorgehensweisen und Technologien, um Projekte in Cloud-Umgebungen durchzuführen. Zum Beispiel:

- Organisatorische Good Practices
- Development- und CI/CD-Umgebungen

LZ 5-2: Application Lifecycle Management in Cloud-Umgebungen umsetzen

Softwarearchitekt:innen kennen Möglichkeiten, um in einer Cloud-Umgebung Deployments und ein Application Lifecycle Management zu realisieren, insbesondere:

- Versionierung von Containern und Deployment-Spezifikationen etc.
- Etablierte Formen der Bereitstellung von Anwendungen, wie z. B.:
  - Rolling Deployment
  - Canary Deployment
  - Blue/Green Deployment

Sie kennen Komponenten und Vorgehensmodelle, um einen schnellen Test- und Deployment-Prozess zu realisieren. Insbesondere Konzepte zum Dev/Test/Prod Cluster wie:

- Verantwortlichkeiten und Zugriffskontrolle
- Good Practices zur Komponentengruppierung

LZ 5-3: Secrets Management Optionen kennen

Softwarearchitekt:innen verstehen die kritische Rolle und Techniken zur Verwaltung von Geheimnissen (Secrets) in der Cloud, darunter:

- Integration von Cloud-Diensten zur Secrets-Verwaltung
- Secrets Operator Konzept

- Automatisierte Secret Rotation

Darüber hinaus können sie die Vor- und Nachteile zum eigenen Betrieb eines Secrets Managers benennen.

5.3. Referenzen

[Cornelia Davis: Cloud Native Patterns], [Ibryam Bilgin, Roland Huss: Kubernetes Patterns.], [Chris Richardson: Microservice Patterns]

6. Automatisierung und Betrieb

Dauer: 150 Min.	Übungszeit: 20 Min.

6.1. Begriffe und Konzepte

DevOps, DevSecOps, Site Reliability Engineering (SRE), Konfiguration, Provisionierung, Infrastructure as Code, Cloud Provider APIs, Berechtigungsverwaltung, Abstraktionsebenen von Container-Managern, Berechnung Verfügbarkeit, Logging, Monitoring, Metriken, Distributed Tracing, Time Series Queries, Alerting, Berechnung Cluster Größe

6.2. Lernziele

LZ 6-1: Neue Rollen und deren Aufgaben kennen und unterscheiden

Softwarearchitekt:innen kennen die neuen Rollen, die im Kontext des Betriebs von Applikationen in der Cloud populär geworden sind, wie DevOps, DevSecOps und SRE.

Sie verstehen die Herausforderungen der Adaption dieser neuen Rollen in klassischen Organisationsstrukturen.

LZ 6-2: Wege zur Schaffung von skalierbaren und hochzuverlässigen Systeme verstehen

Softwarearchitekt:innen verstehen die Möglichkeiten zur Schaffung skalierbarer und hochzuverlässiger Systeme.

Sie kennen die Definition, Methoden und die Herausforderungen des Site Reliability Engineerings.

LZ 6-3: Automatisierungskonzepte zum vorhersehbaren Erzeugen, Verändern und Verbessern der Infrastruktur verstehen

Softwarearchitekt:innen verstehen, dass Automatisierung durch Infrastructure as Code eine Schlüsselmethode modernen Betriebs und eine Komponente des Continuous Delivery ist.

Sie kennen die Möglichkeiten der Automatisierung und verstehen, wie dies mit Werkzeuge zur automatisierten Bereitstellung von Cloud Infrastruktur über die APIs unterschiedlicher Cloud-Anbieter realisiert werden kann.

Sie kennen den Unterschied zwischen Infrastrukturkonfiguration und -Provisionierung, sowie etablierte Vorgehensweisen der Infrastruktur-Verwaltung.

LZ 6-4: Grundprinzipien der Berechtigungsverwaltung verstehen

Softwarearchitekt:innen verstehen die Bedeutung und Implementierung des Prinzips der minimalen Rechtevergabe (Least Privilege) in der Cloud-Umgebung. Sie erkennen die Notwendigkeit, jedem Nutzenden und Service-Account nur die minimalen Berechtigungen zu erteilen, die zur Ausführung ihrer spezifischen Aufgaben notwendig sind.

Dabei können sie folgende Herausforderungen benennen:

- Automatisiertes Ausrollen von Berechtigungseinstellungen

- Verwaltung von konsistenten Berechtigungen in einem Hybrid/Multi-Cloud Setup
- Balance finden zwischen der minimalen Rechtevergabe (Least Privilege) und autonomen Entwicklungsteams
- Durchsetzung von Compliance Vorgaben bei der Berechtigungsverwaltung z.B. über Policy Enforcement

LZ 6-5: Verschiedene Abstraktionsebenen von Container-Managern unterscheiden und verstehen

Softwarearchitekt:innen wissen, dass sich durch Container-Manager die Grundfunktionalität der Container-Orchestrierungswerkzeuge erweitern lassen.

Sie kennen die Einsatzmöglichkeiten vom Container-Managern und können ihre Abstraktionsebenen unterscheiden.

LZ 6-6: Wege der Beobachtbarkeit von verteilten Applikationen kennen

Softwarearchitekt:innen wissen, dass es durch die verteilte Ausführung von Prozessen neue Herausforderungen an die Beobachtbarkeit verteilter Applikationen gibt.

Sie kennen die besonderen Rahmenbedingungen verteilter Anwendung und den Einfluss auf die Beobachtbarkeit mittels:

- Monitoring/Metriken und Alerting
- Logging
- Distributed Tracing

Sie kennen Wege und Verantwortlichkeiten zur Erstellung möglichst fehlervorhersagenden Time Series Queries für Alerts.

LZ 6-7: Berechnungsmethoden zur Dimensionierung von Ressourcen kennen

Softwarearchitekt:innen kennen Methoden zur Berechnung des Ressourcenbedarfs für:

- Verfügbarkeit
- Größe eines Clusters

6.3. Referenzen

[Beyer Betsy et al.: Site Reliability Engineering], [Gene Kim et al.: The Devops Handbook]

7. Case Study

Dauer: 120 Min.	Übungszeit: 120 Min.

Dieser Abschnitt ist nicht prüfungsrelevant.

7.1. Begriffe und Konzepte

Innerhalb einer lehrplankonformen Schulung muss mindestens eine Fallstudie die Konzepte praktisch erläutern.

Art und Ausprägung der vorgestellten Fallstudie kann von der Schulung bzw. den Interessen der Teilnehmenden abhängen.

7.2. Lernziele

LZ 7-1: Vertiefung durch praktische Übungen

Die Fallstudie soll die Themen durch praktische Übungen vertiefen und die Praxis verdeutlichen.

Referenzen

Dieser Abschnitt enthält Ressourcen, die zur Erfüllung des Lehrplans verwendet werden sollten.

- [Awati, R. & Wigmore, I.: Monolithic architecture. WhatIs.com] Awati, R. & Wigmore, I. (2022). Monolithic architecture. WhatIs.com. https://www.techtarget.com/whatis/definition/monolithic-architecture
- [Backend integration. microservice-api-patterns.org] Backend integration. microservice-api-patterns.org. 04.10.2023 retrieved from https://www.microservice-api-patterns.org/patterns/foundation/BackendIntegration
- [Beyer Betsy et al.: Site Reliability Engineering] Beyer Betsy, Chris Jones, Jennifer Petoff, Niall Richard Murphy: Site Reliability Engineering. How Google Runs Production Systems. O'Reilly, 2016
- [Ibryam Bilgin, Roland Huss: Kubernetes Patterns.] Ibryam Bilgin, Roland Huss: Kubernetes Patterns. Reusable Elements for Designing Cloud Native Applications. O'Reilly Media, Second edition, 2023
- [Brendan Burns: Designing Distributed Systems] Brendan Burns: Designing Distributed Systems. Patterns and Paradigms for Scaleable, Reliable Services. O'Reilly Media, 2018
- [Cloud Native landscape] Cloud Native landscape. 04.10.2023 retrieved from https://landscape.cncf.io/card-mode?category=streaming-messaging&grouping=category
- [Cornelia Davis: Cloud Native Patterns] Cornelia Davis: Cloud Native Patterns. Designing Change-tolerant Software. Manning, 2019
- [Dehghani, Z.: How to break a monolith into microservices. martinfowler.com] Dehghani, Z. How to break a monolith into microservices. martinfowler.com. 04.10.2023 retrieved from https://martinfowler.com/articles/break-monolith-into-microservices.html
- [Event sourcing. microservices.io] Event sourcing. microservices.io. 04.10.2023 retrieved from https://microservices.io/patterns/data/event-sourcing.html
- [Frontend Integration. microservice-api-patterns.org] Frontend Integration. microservice-api-patterns.org. 04.10.2023 retrieved from https://www.microservice-api-patterns.org/patterns/foundation/FrontendIntegration
- [Gene Kim et al.: The Devops Handbook] Gene Kim, Jez Humble, Patrick Debois, John Willis, John Allspaw: The Devops Handbook. How to Create World-Class Agility Reliability and Security in Technology Organizations. IT Revolution Press, Second edition, 2021
- [Messaging. microservices.io] Messaging. microservices.io. 04.10.2023 retrieved from https://microservices.io/patterns/communication-style/messaging.html
- [Lewis, J.: Microservices. martinfowler.com] Lewis, J. Microservices. martinfowler.com. 04.10.2023 retrieved from https://martinfowler.com/articles/microservices.html
- [Microservice Architecture Pattern. microservices.io] Microservice Architecture Pattern. microservices.io. 04.10.2023 retrieved from https://microservices.io/patterns/microservices.html
- [Monolithic architecture pattern. microservices.io] Monolithic architecture pattern. microservices.io. 04.10.2023 retrieved from https://microservices.io/patterns/monolithic.html
- [Fowler, M.: MonolithFirst. martinfowler.com] Fowler, M. (2015). MonolithFirst. martinfowler.com. 04.10.2023 retrieved from https://www.martinfowler.com/bliki/MonolithFirst.html
- [Chris Richardson: Microservice Patterns] Chris Richardson: Microservice Patterns. Manning, 2018
- [RSS 2.0 specification (Current)] RSS 2.0 specification (Current). 04.10.2023 retrieved from https://www.rssboard.org/rss-specification

- [SCS: Self-contained systems] SCS: Self-contained systems. 04.10.2023 retrieved from https://scs-architecture.org/
- [Wolff, E.: Self Contained Systems (SCS)] Wolff, E. (2017). Self Contained Systems (SCS): Microservices done right. InfoQ. 04.10.2023 retrieved from https://www.infoq.com/articles/scs-microservices-done-right/