Curriculum for

Certified Professional for Software Architecture (CPSA)®
Advanced Level

Module
CLOUDINFRA

Infrastructure, Containers and Cloud Native

2023.1-RC1-EN-20231124

Table of Contents

List of Learning Goals

Introduction: General information about the iSAQB Advanced Level

What is taught in an Advanced Level module?

- The iSAQB Advanced Level offers modular training in three areas of competence with flexibly designable training paths. It takes individual inclinations and priorities into account.
- The certification is done as an assignment. The assessment and oral exam is conducted by experts appointed by the iSAQB.

What can Advanced Level (CPSA-A) graduates do?

CPSA-A graduates can:

- Independently and methodically design medium to large IT systems
- In IT systems of medium to high criticality, assume technical and content-related responsibility
- Conceptualize, design, and document actions to achieve quality requirements and support development teams in the implementation of these actions
- Control and execute architecture-relevant communication in medium to large development teams

Requirements for CPSA-A certification

- Successful training and certification as a Certified Professional for Software Architecture, Foundation Level® (CPSA-F)
- At least three years of full-time professional experience in the IT sector; collaboration on the design and development of at least two different IT systems
  - Exceptions are allowed on application (e.g., collaboration on open source projects)
- Training and further education within the scope of iSAQB Advanced Level training courses with a minimum of 70 credit points from at least three different areas of competence
- Successful completion of the CPSA-A certification exam

Essentials

What does the module “CLOUDINFRA” convey?

Microservices, containers, and container managers have greatly changed the way we design, develop, and put software into production in recent years. Modern applications must operate in a cluster of multiple nodes, be dynamically placeable, scalable and fault tolerant.

Participants will learn ways to implement dynamic cloud-native architectures, Container Application Design, Logging/Monitoring/Alerting, Container Native Storage and possibilities for UI integration. Likewise, typical concepts of current container managers will be shown and how they can be used to realize common quality requirements for larger web applications. Additionally, common services from various cloud providers will be discussed, possibilities for automation will be shown, software development approaches and application lifecycle will be discussed.

In CLOUDINFRA, the focus is on operational aspects. The concepts of design and implementation of architectures detailed in the FLEX module are explained in overview where necessary for understanding.

Curriculum Structure and Recommended Durations

Content	Recommended minimum duration (min)
1. Fundamentals of Modern Infrastructures	120
2. Common Architecture Concepts	120
3. Cloud Native Journey	240
4. Patterns for Distributed Applications and Cloud Native Architectures	240
5. Development and CI/CD	180
6. Automation and Operation	180
7. Case Study	120
Total	1200 (20h)

Duration, Teaching Method and Further Details

The times stated below are recommendations. The duration of a training course on the CLOUDINFRA module should be at least 3 days, but may be longer. Providers may differ in terms of duration, teaching method, type and structure of the exercises and the detailed course structure. In particular, the curriculum provides no specifications on the nature of the examples and exercises.

Licensed training courses for the CLOUDINFRA module contribute the following credit points towards admission to the final Advanced Level certification exam:

Methodical Competence:	10 Points
Technical Competence:	20 Points
Communicative Competence:	0 Points

Prerequisites

Participants **should** have the following prerequisite knowledge:

- Practical experience in design and development of small to medium sized software systems
- First practical experience in maintenance or further development of software systems
- First practical experience in working with containers and their deployment

Knowledge in the following areas may be **helpful** for understanding some concepts:

- Knowledge or first practical experience in deriving and deploying modern microservice architectures
- First practical experience in dealing with container managers
- First practical experience with common cloud providers

Structure of the Curriculum

The individual sections of the curriculum are described according to the following structure:

- **Terms/principles**: Essential core terms of this topic.
- **Teaching/practice time**: Defines the minimum amount of teaching and practice time that must be spent on this topic or its practice in an accredited training course.
- **Learning goals**: Describes the content to be conveyed including its core terms and principles.

This section therefore also outlines the skills to be acquired in corresponding training courses.

Supplementary Information, Terms, Translations

To the extent necessary for understanding the curriculum, we have added definitions of technical terms to the iSAQB glossary and complemented them by references to (translated) literature.

1. Fundamentals of Modern Infrastructures

Duration: 120 min	Practice time: 0 min

1.1. Terms and Principles

Cloud, cloud types, cloud provider, on premise, bare metal, cloud service models (*aaS), vendor lock-in, managed services, cloud native services, cloud patterns, cloud migration patterns, multi/hybrid cloud, organizational aspects of cloud migration, legal conditions, time-to-market, availability, geo redundancy and scalability, performance, IOPS, decoupling operations, networking.

1.2. Learning Goals

LG 1-1: Understand and Distinguish Cloud Types, Cloud Patterns and Cloud Migration Patterns

Software architects know the common cloud terms and can distinguish between different types such as public, private, hybrid/multi and on-premise.

They understand cloud patterns and cloud migration patterns, and how to apply them.

They understand the difference between cloud and on-premise operations, as well as the arguments for and against cloud operations. They also recognize when the use of own hardware (bare metal) is more appropriate.

LG 1-2: Understand Decision Criteria for the Use of Managed Services and Self-Managed Services

Software architects know different cloud service models (*aaS) and can classify services based on these models.

They understand the shared responsibility model and its relevance for cost and risk assessments when using managed cloud services.

They know the concept of vendor lock-in and its relevance for decision-making between managed and self-managed services.

LG 1-3: Incorporate Compliance and Organizational Aspects Into Cloud Architecture

Software architects are familiar with the organizational aspects of cloud migration and the legal framework for operating applications in the cloud, e.g:

- DSGVO (GDPR)
- GoBD (Principles for electronical accounting and archiving)
- ISO 27001
- IT-Grundschutz (A systematic basis for information security)

They understand that the aspects of data security are relevant for the decision for or against a public cloud and that organizational and/or legal requirements influence the choice of the type of cloud used.

LG 1-4: Apply Availability, Scaling and Performance Requirements as Parameters in Cloud Architecture Design

Software architects understand the impact of the cloud on time-to-market, availability, scaling, performance, IOPS, decoupling operations, and networking.

They can assess when cloud applications require distribution across multiple availability zones and data centers.

Know infrastructure performance requirements and public cloud limitations, such as IOPS for storage.

2. Common Architecture Concepts

Duration: 120 min	Practice time: 20 min

2.1. Terms and Principles

Self-contained Systems, Microservices, independent systems architecture, integration concepts, service discovery, CQRS, Event Sourcing, Eventual Consistency.

2.2. Learning Goals

LG 2-1: Understand Modularization and Communication Options

Software architects are familiar with patterns of modern architectures for distributed applications (especially microservices, self-contained systems) and understand the reasons that lead to these solutions.

- Monolith
- Microsoervices
- Self Contained Systems

They understand the different possibilities of decoupled communication and data exchange between services, for example:

- Event Sourcing
- Messaging Middleware
- HTTP feeds

Software architects understand the implications of a shared-something/nothing architecture, particularly as a result of scaling via containers.

LG 2-2: Understand Modularization and Integration Options

Software architects understand how multiple services can be integrated into an application through different approaches, such as integration via:

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

Software architects understand the impact of eventual consistency on data integrity and consistency.

2.3. References

[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

Duration: 240 min	Practice time: 30 min

3.1. Terms and Principles

Platform quality requirements, Cloud Native Storage, Backup & Restore, Overlay Networking, Network Policies, Container Security, Container Linux.

3.2. Learning Goals

LG 3-1: Understand the Cloud Native Concept

Software architects understand the benefits and goals of the Cloud Native concept. They are familiar with the advantages of the concept, particularly through:

- Unified interfaces for administration and configuration
- Resource abstraction
- Delivery artifacts

LG 3-2: Understand Basic Container Concepts

Software architects understand the technical basics and the added value of containers and know the difference between conventional deployment and deployment with containers.

They understand how the use of containers leads to a better decoupling of responsibilities and thus to a more effective organization.

Software architects understand Linux kernel isolation mechanisms that apply to containers (container security), including:

- Namespaces and cgroups
- Capabilities

They also understand the implications, usage scenarios, and threats of privileged containers.

LG 3-3: Understand Container Manager Abstraction Concepts

Software architects understand the abstraction concepts of current container managers and know best patterns and practices on how to apply quality requirements to them. For example:

- Finite workloads such as (cron) jobs.
- Health Check and Self Healing
- Scaling and load balancing
- Placement

Software architects know the principles of an overlay network (VLAN) for containers and container managers and understand how different services or systems are isolated. They are familiar with requirement scenarios such as implementing a distributed firewall for isolating:

- Multiple tenants
- Test systems in the context of CI/CD

LG 3-4: Understand the Benefits of Specialized Linux Distributions for Operating Containers

Software architects understand how current core/container Linux distributions are optimized for operating containers, the benefits they offer (including kernel and distribution updates), and the implications for the applications being run.

LG 3-5: Evaluate and Select Persistence Solutions in Cloud Environments

Software architects are familiar with the different ways to persist data.

They understand the difference between object and block storage and can classify the different storage services provided by cloud providers, various COTS products, and self-managed solutions for different use cases.

Software architects understand the advantages and disadvantages of a cloud-native storage solution and can compare higher-level managed services (such as RDBMS) provided by cloud providers with the use of self-managed solutions.

They are familiar with various concepts for backing up and restoring persistent data in the cloud.

3.3. References

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

4. Patterns for Distributed Applications and Cloud Native Architectures

Duration: 240 min	Practice time: 20 min

4.1. Terms and Principles

Resilience Patterns, Container Application Design, Cloud Native Architectures, Container Patterns, Service Mesh

4.2. Learning Goals

LG 4-1: Know Different Patterns for Modularizing Cloud-Native Architectures

Software architects are familiar with the concepts of container application design (container pattern) to modularize software components into containers and implement cloud-native architectures.

They are aware of various container application design patterns, such as:

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

Software architects are familiar with methods for separating technical and business tasks through separate containers and understand how container managers handle technical tasks through this principle, particularly:

- Configuration and initialization of applications
- Adaptive scaling using custom metrics
- Container management through operators or controllers

LG 4-2: Ability to Select Appropriate Resilience Patterns to Increase Fault Tolerance

Software architects understand how communication between services in a distributed application can be made fault-tolerant.

They understand which patterns can increase fault tolerance at the communication level and how common service mesh concepts can be used to separate the implementation of resilience patterns from the business code.

Software architects are familiar with the following resilience patterns, among others:

- Circuit Breaker
- Conditional Rate Limits
- Traffic Shifting

They are familiar with the concept of chaos engineering and its methods to test system resilience by deliberately introducing disruptions.

4.3. References

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

5. Development and CI/CD

Duration: 180 min	Practice time: 20 min

5.1. Terms and Principles

Development environment, CI/CD environment, Mean Time To Recovery (MTTR), application lifecycle management, forms of deployments like Rolling-, Canary- and Blue/Green deployment, cluster design, logging, monitoring, metrics, distributed tracing, time series queries, alerting

5.2. Learning Goals

LG 5-1: Implementing Projects in Cloud Environments

Software architects are aware that working in cloud environments brings new requirements to the software development process.

They are familiar with different approaches to implementing projects in cloud environments, such as:

- Organizational good practices
- Development and CI/CD environments

They understand the possibilities of implementing deployments and application lifecycle management in a cloud environment, particularly:

- Versioning of containers and deployment specifications, etc.
- Established forms of application deployment, such as:
  - Rolling deployment
  - Canary deployment
  - Blue/green deployment

They are familiar with components and models for achieving a fast testing and deployment process, particularly concepts related to dev/test/prod clusters, such as:

- Responsibilities and access control
- Good practices for component grouping

LG 5-2: Understanding approaches for observability of distributed applications

Software architects know that the distributed execution of processes presents new challenges for observability of distributed applications.

They understand the unique conditions of distributed applications and their impact on observability through:

- Logging
- Monitoring/metrics and alerting

- Distributed tracing

They are familiar with approaches and responsibilities for creating predictive time series queries for alerts.

5.3. References

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

6. Automation and Operation

Duration: 180 min	Practice time: 20 min

6.1. Terms and Principles

DevOps, DevSecOps, Site Reliability Engineering (SRE), configuration, provisioning, Infrastructure as Code, Cloud Provider APIs, abstraction levels of container manager, calculate availability, calculate cluster size

6.2. Learning Goals

LG 6-1: Understand and Differentiate New Roles and Their Responsibilities

Software architects are familiar with the new roles that have become popular in the context of operating applications in the cloud, such as DevOps, DevSecOps, and SRE.

They understand the challenges of adapting these new roles in traditional organizational structures.

LG 6-2: Understand Ways to Create Scalable and Highly Reliable Systems

Software architects understand the possibilities for creating scalable and highly reliable systems.

They are familiar with the definition, methods, and challenges of Site Reliability Engineering.

LG 6-3: Understand Automation Concepts for Predictable Infrastructure Creation, Modification, and Improvement

Software architects understand that automation through Infrastructure as Code is a key method of modern operations and a component of Continuous Delivery.

They are familiar with the possibilities of automation and understand how this can be realized using tools for automated provisioning of cloud infrastructure through the APIs of different cloud providers.

They understand the difference between infrastructure configuration and provisioning, as well as established practices for infrastructure management.

LG 6-4: Differentiate and Understand Different Abstraction Layers of Container Managers

Software architects are aware that container managers can extend the basic functionality of container orchestration tools.

They are familiar with the use cases for container managers and can differentiate their abstraction layers.

LG 6-5: Understand Resource Sizing Calculation Methods

Software architects are familiar with methods for calculating resource requirements for:

- Availability
- Cluster size

6.3. References

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

7. Case Study

Duration: 120 min	Practice time: 120 min

This section is not examinable.

7.1. Terms and Principles

Within a curriculum-compliant training course, at least one case study must practically explain the concepts.

The type and nature of the case study presented may depend on the training or the interests of the participants.

7.2. Learning Goals

LG 7-1: Deepening the Understanding by Practical Exercises

The Case Study is intended to deepen the topics through practical exercises and to clarify the practice.

References

This section contains resources that should be used to fulfill the curriculum.

- [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/