Syllabus for the Course «Cloud-native Information Systems Architecture»

Course volume: **2 ECTS credits (72 hours)**
Semester: **2**
Certification form:
exam/defense of an architectural project
Class workload:
Lectures - **28 hours**;
Seminars / Digital Labs - **30 hours**
Independent work:
Independent work experience - **14 hours ,**
exam preparation - **32 hours**
Implementation format:
AI- augmented learning (learning using intelligent assistants).

1. General characteristics of the discipline

The discipline **"Architecture of cloud information systems"** refers to the professional module of the master's educational program in the direction **09.04.02 "Information systems and technologies"** .

This course is offered during the second semester of the master's program and develops a systematic understanding of the architectural principles of building modern distributed digital systems designed for operation in a cloud infrastructure.

The discipline considers **cloud-native architecture** not as a set of individual technologies, but as an architectural paradigm for designing information systems, in which:

the infrastructure is considered as a dynamic execution environment;
applications are designed as distributed service systems;
Architectural decisions are made taking into account scalability, resilience, observability, and continuous delivery of changes.

The course is **theoretical and applied engineering in nature** . Its goal is not to study individual containerization or orchestration tools, but to develop students' ability to:

- analyze architectural limitations of distributed systems;
- design the architecture of cloud-native applications;
- justify architectural decisions taking into account scalability and fault tolerance requirements;
- evaluate architectural trade-offs between complexity, performance, and system robustness.

A distinctive feature of the course is the use of **containerization infrastructure and Kubernetes** , studied in parallel as part of the instrumental course, as a laboratory environment for experimental testing of architectural solutions.

Large language models (LLM) are used as a tool for engineering analysis, generation of architectural alternatives and design reflection.

2. The place of the discipline in the structure of the educational program

This course occupies an intermediate position between courses that develop basic skills in the development and infrastructure operation of systems, and courses focused on specialized architectural and platform solutions.
It integrates the knowledge gained in the courses:

- algorithms and data structures;
- databases and information storage systems;
- information systems architecture;
- containerization and Kubernetes ;
- Big Data Engineering.

The discipline provides a transition from **instrumental mastery of infrastructure** to **architectural design of digital systems**.

Within the educational program, the course serves as an **architectural training level** , developing an understanding of how:

- service architecture;
- data architecture;
- execution infrastructure;
- change delivery processes
- form a single engineering system.
- Mastering the discipline creates a methodological basis for subsequent courses and project activities:
- architecture of high-load information systems;
- platform engineering;
- Big Data Engineering;
- digital engineering projects.

3. Objectives of mastering the discipline

The aim of the course is to develop in students a systematic understanding of the architecture of cloud information systems as an engineering discipline for designing distributed service platforms.
Particular attention is paid to:

- understanding the architectural limitations of distributed systems;
- mastering the principles of cloud-native architecture;
- analysis of architectural patterns of service interaction;
- studying the architecture of storage and data consistency;
- mastering architectural mechanisms of sustainability and scalability;
- developing the ability to design system architecture taking into account operational requirements;
- development of skills in engineering argumentation of architectural solutions;
- the formation of a culture of conscious use of LLM in architectural analysis.

4. Objectives of the discipline

The main objectives of the discipline are:

- formation of a systemic understanding of the architecture of distributed information systems;
- studying the principles of cloud-native architecture;
- mastering methods of decomposing complex systems into service components;
- study of service interaction models (synchronous and asynchronous);
- mastering the principles of data management in distributed systems;
- study of data consistency mechanisms and distributed transactions;
- mastering architectural patterns of sustainability;
- studying the principles of scaling service systems;
- mastering the principles of observability and diagnostics of distributed systems;
- development of skills in architectural analysis and audit of information systems;
- development of skills for experimental verification of architectural solutions;
- development of a culture of critical use of LLM in architectural design.

5. Planned learning outcomes

As a result of mastering the discipline, the student must:

Know:

- architectural principles of cloud-native systems;
- architectural limitations of distributed computing environments;
- main architectural styles of distributed systems;
- service interaction patterns;
- data consistency models;
- architectural patterns of sustainability and scalability;
- principles of observability and operational diagnostics;
- The main architectural risks of distributed systems.

Be able to:

- analyze the requirements for the architecture of a digital system;
- perform decomposition of the system into service components;
- design the architecture of service interactions;
- determine the architecture of data storage and processing;
- choose architectural patterns for sustainability and scalability;
- analyze architectural trade-offs;
- perform an architectural audit of the information system;
- use LLM as a tool for analyzing architectural solutions.

Be proficient in:

- methods of architectural analysis of distributed systems;
- skills in system design of cloud-native architectures;
- skills for reasoned selection of architectural solutions;
- methods of comparative analysis of architectural alternatives;
- a culture of engineering reflection and critical evaluation of LLM recommendations.

6. Methodological concept of the discipline

6.1 Architecture as a system level of a digital platform

A modern information system is a multi-layered architectural structure that includes:

- application services layer;
- data management level;
- execution infrastructure level;
- observability and exploitation contour.

Within the discipline, architecture is viewed as **a mechanism for reconciling requirements and constraints** that arise during the design of distributed systems.

6.2 Architectural solutions as a space of compromises

The architecture of cloud-native systems is always formed under engineering constraints:

- network delays;
- partial component failures;
- data consistency constraints;
- load variability;
- complexity of system maintenance.

Therefore, architectural design is seen as a process **of making trade-offs** between:

- scalability;
- stability;
- productivity;
- ease of maintenance.

The goal of the discipline is to develop the ability to make such decisions consciously.

6.3 Experimental verification of architectural solutions

One of the key features of the course is the use of an experimental approach.
Architectural hypotheses are tested through:

- deploying services in a container environment;
- modeling of component interactions;
- load experiments;
- failure injection;
- observability metrics analysis.

Thus, architecture is studied not only as a theory, but also as **a practice of engineering experimentation**.

6.4 Architectural project as an integrated form of training

The course includes **a comprehensive architectural project** completed over the course of a semester.
The project involves:
selection of the subject area of the digital service;
system architecture design;

justification of architectural solutions;
experimental verification of key architectural hypotheses;
architectural audit of the developed solution.
This project serves as an integration of theoretical knowledge and practical skills.

7. Using LLM in the educational process

Large language models are used as a tool to support architectural analysis.
LLMs are applied in several modes.

LLM as a reference tool

The model is used to explain the terms of distributed systems architecture, service interaction patterns, and the architectural principles of cloud-native systems.

LLM as an intelligent assistant

LLM helps students:

- analyze architectural requirements;
- generate alternative architectural solutions;
- identify potential architectural risks;
- formulate arguments for choosing architectural solutions.

LLM as an architectural opponent

LLM is used for:

- critical analysis of architectural schemes;
- identifying weaknesses in the architecture;
- generating of alternative architectural options;
- formation of clarifying questions.

Critical Considerations: Limitations of LLM in Architectural Analysis

The use of LLM in architectural design requires caution.
Typical errors of models:

- proposing standard architectural patterns without analyzing requirements;
- ignoring infrastructure limitations;
- a superficial analysis of architectural compromises;
- enumeration of patterns without engineering choice.

Critical verification of LLM recommendations is a mandatory element of independent work.

8. Educational technologies

The discipline uses an AI-augmented learning model, which combines traditional forms of learning with the use of intelligent assistants.
The educational process includes:

- lecture classes with architectural analysis;
- seminar classes;
- digital laboratory work;
- engineering discussions;
- architectural workshop;

- implementation of an end-to-end architectural project;
- use LLM for analysis and project reflection.

9. Organization of independent work of students

Students' independent work (SDS – Self-Directed Studies) is aimed at developing skills in architectural analysis and design of distributed systems.
The SDS includes:

- analysis of architectural cases;
- preparation of architectural schemes;
- development of architectural notes;
- completing laboratory assignments;
- preparation of analytical materials;
- implementation of stages of an end-to-end architectural project;
- work with LLM assistants with mandatory engineering verification of results.

Independent work is structured along two lines:

- regular thematic assignments;
- stages of an end-to-end architectural project.

10. Forms of control

The following forms of control are used within the framework of the discipline.
Current assessment

- discussion of architectural cases;
- performing laboratory work;
- quick knowledge checks;
- evaluation of architectural schemes and analytical reports;
- evaluation of the stages of an end-to-end project.

Interim assessment
defense of an architectural project..
Final assessment
exam including:

- architectural case analysis;
- selection of architectural solutions;
- justification of architectural trade-offs;
- oral defense of the proposed solution.

The LLM can be used to generate cases and analyse written work, but the final grade is determined by the instructor.

Curriculum schedule for the course

Key:

- **Lecture (Lect.)** — theoretical material delivery.
- **Practical / Lab.** — hands-on activities, simulations and discussions.
- **Self-directed work (SDS)** — independent study on project tasks and analysis.

- **LLM use** — applying Large Language Models for analysis, generation and decision support (integrated into independent study tasks).

Week	Content	Lect. (h)	Practice (h)	SDS (h)
1	**Lecture.** Introduction to cloud information system architecture. The evolution of architecture: monolithic systems, distributed systems, and cloud computing. Limitations of distributed systems and the role of cloud infrastructure in scaling services. **Seminar / digital lab / discussion.** Analysis of the architecture of large digital platforms (Netflix , Amazon, Uber). Discussion of architectural solutions that ensure scalability and resilience. **Quick knowledge check.** Understanding the differences between monolithic and distributed architecture, identifying the key properties of cloud-native systems. **SDS - regular homework.** Prepare a comparative table of architectural models: monolith, distributed system, cloud-native system. **SDS - end-to-end project stage.** Selecting the subject area of the project and formulating system requirements.	2	2	4
2	**Lecture.** Cloud-native architecture principles : scalability, elasticity, stateless approach, immutable infrastructure. 12-Factor App concept . **Seminar / digital lab.** Analysis of the microservice system architecture and identification of cloud-native principles. **Quick knowledge check.** Definition of the architectural properties of cloud-native systems and explanation of their role. **SDS — regular homework.** Analysis of the architecture of an existing cloud service. **SDS — project stage.** Formulation of the system's architectural requirements (scalability, fault tolerance, observability).	2	2	4
3	**Lecture.** Decomposition of information systems. Domain-Driven Design, bounded Context , defining service boundaries. **Seminar/digital lab.** Decomposition of the subject area of the selected service. **Quick knowledge check.** Defining service boundaries and analyzing the architectural decomposition. **Regular homework assignment.** Preparation of an architectural decomposition diagram of the system.	2	2	4

Week	Content	Lect. (h)	Practice (h)	SDS (h)
	Regular homework assignment. Defining services and their functional responsibilities.			
4	**Lecture.** Synchronous service interactions: REST, gRPC , API contracts. **Seminar/lab.** Designing API interactions between services. **Quick knowledge check.** Understanding the differences between REST and gRPC , and the principles of API contracts. **Regular homework.** Prepare an API interaction specification. **SDS is a project stage.** Designing an API system.	2	2	4
5	**Lecture.** Asynchronous architecture. Event- driven systems , publish-subscribe , message brokers. **Seminar/laboratory.** Designing an event-driven system architecture. **Quick knowledge check:** Explaining the benefits of asynchronous architecture. **SDS - regular homework.** Develop an event-driven architecture diagram. **SDS is a project phase.** Defining events and message flows.	2	2	4
6	**Lecture :** API Gateway and Service Discovery. Integration Layer Architecture. **Seminar.** Analysis of Integration Services Architecture. **Quick Knowledge Check.** API Gateway Functions. **Regular homework.** Prepare the integration layer architecture. **SDS is a project stage.** Adding a gateway layer to the project architecture.	2	2	4
7	**Lecture .** Architecture data : database-per-service, polyglot persistence. **Seminar.** Analysis of data storage architecture. **Quick knowledge check.** Benefits of database-per-service. **SDS - regular homework.** Preparation of data storage architecture. **SDS is a project stage.** Designing service data models.	2	2	4
8	**Lecture .** Coherence data : distributed transactions, eventual consistency, Saga. **Seminar/lab.** Analysis of coordination scenarios. **Quick knowledge check.** Understanding the Saga pattern. **RDS — regular homework.** Description of the data consistency scenario. **The SDS is a project stage.** Designing a coordination mechanism.	2	2	4
9	**Lecture.** Limitations of distributed systems: latency , partial failure , CAP theorem. **Seminar.** Analysis of architectural risks in distributed systems. **Quick knowledge check.** Interpretation of the CAP theorem. **RDS — regular homework.** Architectural risk analysis. **SDS is a project stage.** Identifying project architecture risks.	2	2	4
10	**Lecture:** Fault-tolerance patterns: retry , circuit breaker , bulkhead . **Seminar/lab.** Analysis of service failure scenarios.	2	2	4

Week	Content	Lect. (h)	Practice (h)	SDS (h)
	Quick knowledge check. Mechanisms for preventing cascading failures. **Regular homework. Preparing** a system resilience strategy. **SDS is the project stage.** Adding sustainability mechanisms.			
11	**Lecture.** Scalability architecture: horizontal scaling, load balancing, caching. **Seminar/lab.** Analysis of scalable systems architecture. **Quick knowledge check:** Principles of horizontal scaling. **Regular homework assignments.** Develop a scaling strategy. **SDS is a project stage.** Designing a scaling architecture.	2	2	4
12	**Lecture .** Observability: logging, metrics, distributed tracing. **Seminar/lab.** Analysis of system metrics. **Quick knowledge check.** Observability principles. **Regular homework.** Develop a monitoring architecture. **SDS is a project stage.** Adding observability to the architecture.	2	2	4
13	**Lecture.** Change delivery architecture: CI/CD, update strategies (blue-green , canary). **Seminar.** Pipeline architecture analysis . **Quick knowledge check.** Understanding deployment strategies. **Regular homework assignments.** CI/CD architecture preparation. **SDS is a project stage.** Designing a system update strategy.	2	2	4
14	**Lecture .** Platform engineering. Internal Developer Platform, multi-tenant architecture . **Seminar.** Analysis of cloud platform architecture. **Quick Knowledge Check:** The Role of Platform Architecture. **Regular homework assignments.** Prepare an analysis of the platform solution. **The SDS is a project stage** that involves developing the complete system architecture.	2	2	4
15	**Lecture.** Methods of architectural audit of information systems. **Seminar/laboratory.** Engineering audit of project architecture. **Quick knowledge check.** Identifying architectural risks. **SDS — regular homework.** Preparation of the final architectural report. **SDS — project stage.** Completion of the architectural design.	0	2	6
16	**Exam:** Architectural case analysis and architectural project defense.	-	4	32