



National University of Engineering (UNI)
School of Computer Science
Syllabus 2027-I

1. COURSE

CS394. Software Architecture and DevOps for Cloud Software (Elective)

2. GENERAL INFORMATION

2.1 Course	: CS394. Software Architecture and DevOps for Cloud Software
2.2 Semester	: 8 th Semester.
2.3 Credits	: 3
2.4 Hours	: 2 HT; 2 HP;
2.5 Duration of the period	: 16 weeks
2.6 Type of course	: Elective
2.7 Learning modality	: Face to face
2.8 Prerequisites	: CS292. Software Engineering II. (7 th Sem)

3. PROFESSORS

Meetings after coordination with the professor

4. INTRODUCTION TO THE COURSE

Modern software design requires combining solid architectural principles with agile continuous delivery practices and cloud operations. This course integrates three areas that are inseparable in today's industry: software architecture, DevOps, and cloud computing.

Students will learn to design systems at scale using well-established architectural patterns, to automate build, test, and deployment cycles through CI/CD pipelines, and to provision and operate infrastructure on public cloud platforms using containers, orchestration, and infrastructure-as-code tools.

The course places special emphasis on the reliability and observability of distributed systems, covering metrics, distributed traces, error budgets, and site reliability engineering (SRE) practices as applied in high-performing technology organizations.

5. GOALS

- Design scalable and maintainable software systems by applying modern architectural patterns.
- Implement continuous integration and continuous delivery (CI/CD) pipelines to accelerate and secure the software lifecycle.
- Deploy and manage containerized applications on public cloud platforms using industry-standard tools.
- Provision cloud infrastructure in a reproducible and versioned manner through infrastructure as code (IaC).
- Apply observability and SRE practices to ensure the reliability of systems in production.

6. COMPETENCES

AG-C09) Design and Development of Solutions: Designs, implements, and evaluates solutions for complex computing problems. (Assessment)

AG-C11) Use of Tools: Applies modern computing tools in problem solving. (Assessment)

AG-C12) Uses systemic approaches to select, develop, apply, integrate, and manage secure computing technologies to achieve user objectives. (Usage)

7. TOPICS

Unit 1: Principles and Software Architecture (8 hours)	
Competences Expected: AG-C09,AG-C12	
Topics	Learning Outcomes
• System design principles. System Reliability – Levels of abstraction (e.g., architectural design and detailed design) – Separation of concerns – Information hiding – Coupling and cohesion • Software architecture. System Reliability – Design paradigms * Top-down functional decomposition/layered design * Data-oriented architecture * Object-oriented analysis and design * Event-driven design – Standard architectures (e.g., client-server and microservice architectures including REST discussions, n-layer, pipes-and-filters, Model View Controller) – Identifying component boundaries and dependencies • Programming in the large vs programming in the small. System Reliability • Structural and behavioral models of software designs • Design modeling, for instance with class diagrams, entity relationship diagrams, or sequence diagrams • API design principles – Consistency * Consistent APIs are easier to learn and less error-prone * Consistency is both internal (between different portions of the API) and external (following common API patterns) – Composability – Documenting contracts * API operations should describe their effect on the system, but not generally their implementation * Preconditions, postconditions, and invariants – Expandability – Error reporting * Errors should be clear, predictable, and actionable * Input that does not match the contract should produce an error * Errors that can be reliably managed without reporting should be managed	• Identify the standard software architecture of a given high-level design [Analyze] • Identify the dependencies among a set of software components in an architectural design [Analyze] • Select and use an appropriate design paradigm to design a simple software system and explain how system design principles have been applied in this design [Evaluate] • Translate a natural language software design into class diagrams [Translate]

Unit 2: Software Design Quality and Evaluation (8 hours)	
Competences Expected: AG-C09,AG-C12	
Topics	Learning Outcomes
• Evaluating design tradeoffs (e.g., efficiency vs reliability, security vs usability) • Architectural styles: layered, event-driven, microkernel, service-oriented, pipeline. • Quality attributes: availability, scalability, performance, security, maintainability. • Requirement traceability – Understanding which requirements are satisfied by a design • Measurement and analysis of design quality	• Contrast two software designs across different qualities, such as efficiency or usability [Contrast] • Identify which requirements are satisfied by a provided software design [Analyze] • Adapt a flawed system design to better follow principles such as separation of concerns or information hiding [Create] • Select the most appropriate architectural style given the quality attributes required by a system. [Evaluate]
Readings : [Richards2020]	

Unit 3: API Design and Modeling (6 hours)	
Competences Expected: AG-C09,AG-C11	
Topics	Learning Outcomes
• Microservices principles: cohesion, loose coupling, and service boundaries. • Monolith decomposition: domain-driven design (DDD). • API design principles – Consistency * Consistent APIs are easier to learn and less error-prone * Consistency is both internal (between different portions of the API) and external (following common API patterns) – Composability – Documenting contracts * API operations should describe their effect on the system, but not generally their implementation * Preconditions, postconditions, and invariants – Expandability – Error reporting * Errors should be clear, predictable, and actionable * Input that does not match the contract should produce an error * Errors that can be reliably managed without reporting should be managed • API Gateway and service mesh. • Resilience patterns: Circuit Breaker, Retry, Bulkhead, Timeout. • Data Stores – Cooperatively maintained data structures implementing maps and related ADTs – Varieties: Owned, shared, sharded, replicated, immutable, versioned	• Design an API for a single component of a large software system, including identifying and documenting each operation's invariants, contract, and error conditions [Design] • Evaluate an API description in terms of consistency, composability, and expandability [Evaluate] • Design a microservices-based system by decomposing a monolithic application into independent services. [Usar] • Apply resilience patterns to tolerate failures in distributed systems. [Usar]
Readings : [Newman2021]	

Unit 4: Virtualization (6 hours)	
Competences Expected: AG-C11,AG-C12	
Topics	Learning Outcomes
- Containers and their comparison to virtual machines. - Types of virtualizations (including hardware/software, OS, server, service, network). See also: Latency, Cache and Memory Hierarchy, Virtualization and Isolation - Portable virtualization; emulation vs isolation. See also: Latency, Cache and Memory Hierarchy, Virtualization and Isolation - Virtual machines and container escapes, dangers from a security perspective. See also: Security Design and Controls Engineering, Threat Analysis and Security Engineering, Trusted Computing and Privacy Engineering - Dockerfile: best practices for building efficient and secure images. - Multi-container application composition with docker-compose.	- Explain the difference between emulation and isolation [Explain] - Evaluate virtualization tradeoffs [Evaluate] - Build and deploy containerized applications using Docker. [Usar] - Apply security best practices when building Docker images. [Usar]
Readings : [Burns2022]	

Unit 5: Coordination (6 hours)	
Competences Expected: AG-C09,AG-C11	
Topics	Learning Outcomes
- Kubernetes architecture: control plane and worker nodes. - Core objects: Pods, Deployments, Services, ConfigMaps, Secrets. - Horizontal Pod Autoscaling (HPA) and self-healing. - Dependencies - Initiation or progress of one activity may be dependent on other activities, so as to avoid race conditions, ensure termination, or meet other requirements - Ensuring progress by avoiding dependency cycles, using monotonic conditions, removing inessential dependencies - Package management with Helm.	- Deploy and manage containerized applications on a Kubernetes cluster. [Usar] - Configure auto-scaling and self-healing for cloud-native workloads. [Usar] - Modify a concurrent system to use a more scalable, reliable, or available data store [Create]
Readings : [Burns2022]	

Unit 6: Version Control and CI/CD (6 hours)	
Competences Expected: AG-C11,AG-C12	
Topics	Learning Outcomes
• Software configuration management and version control: Software Development Practices – Configuration in version control, reproducible builds/configuration. – Version control branching strategies. Development branches vs release branches. Trunk-based development. – Merging/rebasing strategies, when relevant. • Release management. • Software process automation: – Build systems - the value of fast, hermetic, reproducible builds, compare/contrast approaches to building a project. – Continuous Integration (CI) - the use of automation and automated tests to do preliminary validation that the current head/trunk revision builds and passes (basic) tests. – Continuous Deployment (CD) - the use of automation to automatically release every change that passes the automated tests to the production environment, ensuring frequent and reliable deliveries. – Dependency management - updating external/upstream dependencies, package management, SemVer. • Testing tools including static and dynamic analysis tools. Software Development Practices, Information Flow and Non-Interference, Injection and Input Validation, Memory Safety and Types, Malware Analysis and Advanced Security • Tool integration concepts and mechanisms. Software Development Practices • DevOps culture: flow, feedback, and continuous learning.	• Describe the difference between centralized and distributed software configuration management [Describe] • Describe how version control can be used to help manage software release management [Describe] • Understand the use of CI/CD systems as a ground-truth for the state of the team's shared code (build and test success) [Explain] • Demonstrate the capability to use software tools in support of the development of a software product of medium size [Demonstrate]
Readings : [Kim2021]	

Unit 7: Version Control and CI/CD (6 hours)	
Competences Expected: AG-C11,AG-C12	
Topics	Learning Outcomes
• IaC principles: reproducibility, idempotency, and infrastructure versioning. • Declarative vs. imperative approaches. • Terraform: providers, resources, modules, and state management. • Cloud-native IaC tools: AWS CloudFormation, Azure ARM Templates, GCP Deployment Manager. • Software configuration management and version control: Software Development Practices – Configuration in version control, reproducible builds/configuration. – Version control branching strategies. Development branches vs release branches. Trunk-based development. – Merging/rebasing strategies, when relevant.	• Write and apply Terraform configurations to provision cloud infrastructure. [Usar] • Identify configuration items and use a source code control tool in a small team-based project [Analyze] • Compare IaC tools based on their state management and modularity capabilities. [Evaluar]
Readings : [Morris2021], [Kim2021]	

Unit 8: Networked Applications (6 hours)	
Competences Expected: AG-C09,AG-C11	
Topics	Learning Outcomes
• Cloud service models: IaaS, PaaS, SaaS, FaaS. • Distributed application paradigms (e.g., client/server, peer-to-peer, cloud, edge, and fog). See also: Communication, Coordination • Major providers: AWS, Google Cloud, Azure — core services. • Compute services: virtual machines, serverless functions, managed containers. • Storage services: object, block, and managed databases. • Cloud networking: VPCs, load balancers, CDN.	• Analyze the needs of specific networked application demands [Analyze] • Compare IaaS, PaaS, SaaS, and FaaS models to select the most appropriate one for a given architecture. [Evaluar] • Design a cloud-native application using managed services from a major cloud provider. [Usar]
Readings : [Kim2021], [Burns2022]	

Unit 9: Reliability Engineering (6 hours)	
Competences Expected: AG-C11,AG-C12	
Topics	Learning Outcomes
• Measurement-based analysis of software reliability - telemetry, monitoring and alerting, dashboards, release qualification metrics, etc. • The three pillars of observability: metrics, logs, and distributed traces. • Distributed tracing and correlation identifiers. • Software reliability engineering practices - including reviews, testing, practical model checking • Alerting strategies and on-call practices.	• Describe how to determine the level of reliability required by a software system [Describe] • Identify methods that will lead to the realization of a software architecture that achieves a specified level of reliability [Analyze] • Implement observability in a cloud application using metrics, logs, and distributed traces. [Usar]
Readings : [Kim2021]	

Unit 10: System Reliability (6 hours)	
Competences Expected: AG-C09,AG-C12	
Topics	Learning Outcomes
• Concept of reliability as probability of failure or mean time between failures, and faults as cause of failures • Compare the "error budget" approach to reliability with the "error-free" approach and identify domains where each is relevant. • Software fault tolerance techniques and models – Contextual differences in fault tolerance (e.g., crashing a flight critical system is strongly avoided, crashing a data processing system before corrupt data is written to storage is highly valuable) • Reliability through redundancy: – check and retry Reliability Support, Fault tolerance – redundant encoding (error correction codes, CRC, FEC, RAID) Memory Hierarchy, Distributed Databases/Cloud Computing, Reliability Support – duplication/mirroring/replicas Distributed Databases/Cloud Computing • Software reliability, system reliability and failure behavior • Service Level Indicators (SLI), Objectives (SLO), and Agreements (SLA).	• Identify ways to apply redundancy to achieve fault tolerance [Analyze] • Identify single-point-of-failure (SPF) dependencies in a system design [Analyze] • Explain the problems that exist in achieving very high levels of reliability [Explain] • Define SLIs, SLOs, and error budgets for a service in production. [Usar]
Readings : [Kim2021]	

8. WORKPLAN

8.1 Methodology

Individual and team participation is encouraged to present their ideas, motivating them with additional points in the different stages of the course evaluation.

8.2 Theory Sessions

The theory sessions are held in master classes with activities including active learning and roleplay to allow students to internalize the concepts.

8.3 Practical Sessions

The practical sessions are held in class where a series of exercises and/or practical concepts are developed through problem solving, problem solving, specific exercises and/or in application contexts.

9. EVALUATION SYSTEM

Theory Sessions:

The theory sessions are held in master classes with activities including active learning and roleplay to allow students to internalize the concepts.

Lab Sessions:

In order to verify their competences, several activities including active learning and roleplay will be developed during lab sessions.

Oral Presentations:

Individual and team participation is encouraged to present their ideas, motivating them with additional points in the different stages of the course evaluation.

Reading:

Throughout the course different readings are provided, which are evaluated. The average of the notes in the readings is considered as the mark of a qualified practice. The use of our virtual campus allows each student to access the course information, and interact outside the classroom with the teacher and with the other students.

The final grade NF depends on the following grades:

- PE : Partial Exam.
- FE : Final Exam.
- P : Permanent Evaluation (Average of practical work and graded assignments).

The final grade NF is calculated as follows:

$$NF = \frac{PE + FE + P}{3}$$

10. BASIC BIBLIOGRAPHY