Universidad Nacional de Ingeniería (UNI)
Escuela Profesional de
Ciencia de la Computación
Sílabo 2026-I

1. CURSO

CS394. Arquitectura de Software y DevOps para Software en la Nube (Electivo)

2. INFORMACIÓN GENERAL

2.1 Curso	:	CS394. Arquitectura de Software y DevOps para Software en la Nube
2.2 Semestre	:	8^{vo} Semestre.
2.3 Créditos	:	3
2.4 Horas	:	2 HT; 2 HP;
2.5 Duración del periodo	:	16 semanas
2.6 Condición	:	Electivo
2.7 Modalidad de aprendizaje	:	Presencial
2.8 Prerrequisitos	:	CS292. Ingeniería de Software II. (7^{mo} Sem)

3. PROFESORES

Atención previa coordinación con el profesor

4. INTRODUCCIÓN AL CURSO

El diseño de software moderno exige combinar principios sólidos de arquitectura con prácticas ágiles de entrega continua y operación en la nube. Este curso integra tres áreas que en la industria actual son inseparables: arquitectura de software, DevOps y computación en la nube.

El estudiante aprenderá a diseñar sistemas a escala usando patrones arquitectónicos reconocidos, a automatizar los ciclos de construcción, prueba y despliegue mediante pipelines de CI/CD, y a provisionar y operar infraestructura en plataformas de nube pública mediante contenedores, orquestación y herramientas de infraestructura como código.

El curso pone especial énfasis en la confiabilidad y observabilidad de sistemas distribuidos, cubriendo métricas, trazas distribuidas, presupuestos de error y prácticas de ingeniería de confiabilidad del sitio (SRE), tal como se aplican en organizaciones de tecnología de alto rendimiento.

5. OBJETIVOS

- Diseñar sistemas de software escalables y mantenibles aplicando patrones arquitectónicos modernos.
- Implementar pipelines de integración y entrega continua (CI/CD) para acelerar y asegurar el ciclo de vida del software.
- Desplegar y gestionar aplicaciones en contenedores sobre plataformas de nube pública usando herramientas estándar de la industria.
- Provisionar infraestructura de nube de forma reproducible y versionada mediante infraestructura como código (IaC).
- Aplicar prácticas de observabilidad y SRE para garantizar la confiabilidad de sistemas en producción.

6. RESULTADOS DEL ESTUDIANTE

AG-C09) Diseño y Desarrollo de Soluciones: Diseña, implementa y evalúa soluciones para problemas complejos de computación. (Assessment)

AG-C11) Uso de Herramientas: Aplica herramientas modernas de computación en la resolución de problemas. (Assessment)

AG-C12) Usa enfoques sistémicos para seleccionar, desarrollar, aplica, integrar y administrar tecnologías seguras de computación para lograr los objetivos del usuario. (Usage)

Unidad 1: Principios y Arquitectura de Software (8 horas)	
Resultados esperados: AG-C09,AG-C12	
Temas	**Aprendizaje esperado (*Learning Outcomes*)**
• Principios de diseño de sistemas. Fundamentos de Sistemas (SF)-Reliability – Niveles de abstracción (por ejemplo, diseño arquitectónico y diseño detallado) – Separación de preocupaciones (separation of concerns) – Ocultación de información (information hiding) – Acoplamiento y cohesión (coupling and cohesion) • Arquitectura de software. Fundamentos de Sistemas (SF)-Reliability – Paradigmas de diseño ∗ Descomposición funcional de arriba hacia abajo/diseño por capas ∗ Arquitectura orientada a datos ∗ Análisis y diseño orientado a objetos ∗ Diseño dirigido por eventos – Arquitecturas estándar (por ejemplo, cliente-servidor y arquitecturas de microservicios incluyendo discusiones sobre REST, n-capas, tuberías y filtros, Modelo Vista Controlador) – Identificar límites y dependencias de componentes • Programación a gran escala vs programación a pequeña escala (programming in the large vs programming in the small). Fundamentos de Sistemas (SF)-Reliability • Modelos estructurales y de comportamiento de diseños de software • Modelado de diseño, por ejemplo con diagramas de clases, diagramas de entidad-relación o diagramas de secuencia • Principios de diseño de API – Consistencia ∗ Las API consistentes son más fáciles de aprender y menos propensas a errores ∗ La consistencia es tanto interna (entre diferentes partes de la API) como externa (siguiendo patrones comunes de API) – Capacidad de composición (*composability*) – Documentación de contratos ∗ Las operaciones de la API deben describir su efecto en el sistema, pero generalmente no su implementación ∗ Precondiciones, postcondiciones e invariantes – Capacidad de expansión (*expandability*) – Reporte de errores	• Identificar la arquitectura de software estándar de un diseño de alto nivel dado [Analizar] • Identificar las dependencias entre un conjunto de componentes de software en un diseño arquitectónico [Analizar] • Seleccionar y usar un paradigma de diseño apropiado para diseñar un sistema de software simple y explicar cómo se han aplicado los principios de diseño de sistemas en este diseño [Evaluar] • Traducir un diseño de software en lenguaje natural a diagramas de clases [Traducir]

Unidad 2: Calidad y Evaluación del Diseño (8 horas)	
Resultados esperados: AG-C09,AG-C12	
Temas	**Aprendizaje esperado (*Learning Outcomes*)**
• Evaluar compensaciones de diseño (tradeoffs) (por ejemplo, eficiencia vs confiabilidad, seguridad vs usabilidad) • Estilos arquitectónicos: en capas, orientado a eventos, microkernel, orientado a servicios, pipeline. • Atributos de calidad: disponibilidad, escalabilidad, rendimiento, seguridad, mantenibilidad. • Trazabilidad de requisitos – Comprender qué requisitos son satisfechos por un diseño • Medición y análisis de la calidad del diseño	• Contrastar dos diseños de software a través de diferentes cualidades, como eficiencia o usabilidad [Contrastar] • Identificar qué requisitos son satisfechos por un diseño de software proporcionado [Analizar] • Adaptar un diseño de sistema defectuoso para que siga mejor principios como la separación de preocupaciones o la ocultación de información [Crear] • Seleccionar el estilo arquitectónico más adecuado dados los atributos de calidad requeridos por un sistema. [Evaluar]
Lecturas : [RF20]	

Unidad 3: Diseño de API y Modelado (6 horas)

Resultados esperados: AG-C09,AG-C11

Temas	**Aprendizaje esperado (*Learning Outcomes*)**
• Principios de microservicios: cohesión, acoplamiento débil y fronteras de servicio. • Descomposición de monolitos: diseño orientado al dominio (DDD). • Principios de diseño de API – Consistencia ∗ Las API consistentes son más fáciles de aprender y menos propensas a errores ∗ La consistencia es tanto interna (entre diferentes partes de la API) como externa (siguiendo patrones comunes de API) – Capacidad de composición (*composability*) – Documentación de contratos ∗ Las operaciones de la API deben describir su efecto en el sistema, pero generalmente no su implementación ∗ Precondiciones, postcondiciones e invariantes – Capacidad de expansión (*expandability*) – Reporte de errores ∗ Los errores deben ser claros, predecibles y accionables ∗ La entrada que no coincida con el contrato debe producir un error ∗ Los errores que pueden gestionarse de manera confiable sin reportarse deben gestionarse • API Gateway y malla de servicios (*service mesh*). • Patrones de resiliencia: Circuit Breaker, Retry, Bulkhead, Timeout. • Almacenes de Datos – Estructuras de datos mantenidas cooperativamente que implementan mapas y TADs relacionados. – Variedades: Propiedad exclusiva, compartida, fragmentada, replicada, inmutable, versionada.	• Diseñar una API para un solo componente de un sistema de software grande, incluyendo la identificación y documentación de las invariantes, el contrato y las condiciones de error de cada operación [Diseñar] • Evaluar una descripción de API en términos de consistencia, capacidad de composición y capacidad de expansión [Evaluar] • Diseñar un sistema basado en microservicios descomponiendo una aplicación monolítica en servicios independientes. [Usar] • Aplicar patrones de resiliencia para tolerar fallos en sistemas distribuidos. [Usar]

Lecturas : [New21]

Unidad 4: Virtualización (6 horas)	
Resultados esperados: AG-C11,AG-C12	
Temas	**Aprendizaje esperado (*Learning Outcomes*)**
• Contenedores y su comparación con máquinas virtuales. • Tipos de virtualización (incluyendo hardware/software, SO, servidor, servicio, red). Ver también: Fundamentos de Sistemas (SF)-Performance • Virtualización portátil; emulación vs aislamiento. Ver también: Fundamentos de Sistemas (SF)-Performance • Máquinas virtuales y escapes de contenedores, peligros desde una perspectiva de seguridad. Ver también: Seguridad (SEC)-Engineering • Dockerfile: buenas prácticas para construir imágenes eficientes y seguras. • Composición de aplicaciones multi-contenedor con docker-compose.	• Explicar la diferencia entre emulación y aislamiento [Explicar] • Evaluar las compensaciones de la virtualización [Evaluar] • Construir y desplegar aplicaciones en contenedores usando Docker. [Usar] • Aplicar buenas prácticas de seguridad en la construcción de imágenes Docker. [Usar]
Lecturas : [Bur+22]	

Unidad 5: Coordinación (6 horas)	
Resultados esperados: AG-C09,AG-C11	
Temas	**Aprendizaje esperado (*Learning Outcomes*)**

• Arquitectura de Kubernetes: plano de control y nodos trabajadores.
• Objetos fundamentales: Pods, Deployments, Services, ConfigMaps, Secrets.
• Escalado horizontal automático (*HPA*) y autocuración.
• Dependencias
  – La iniciación o progreso de una actividad puede depender de otras actividades, para evitar condiciones de carrera, garantizar la terminación o cumplir otros requisitos.
  – Garantizar el progreso evitando ciclos de dependencia, usando condiciones monótonas, eliminando dependencias no esenciales.
• Gestión de paquetes con Helm.
	• Desplegar y gestionar aplicaciones en contenedores sobre un clúster de Kubernetes. [Usar]
	• Configurar el escalado automático y la autocuración para cargas de trabajo nativas de la nube. [Usar]
	• Modificar un sistema concurrente para usar un almacén de datos más escalable, confiable o disponible [Crear]

Lecturas : [Bur+22]	

Unidad 6: Control de Versiones y CI/CD (6 horas)

Resultados esperados: AG-C11,AG-C12

Temas	**Aprendizaje esperado (*Learning Outcomes*)**
• Gestión de configuración de software y control de versiones: Fundamentos de Desarrollo de Software (SDF)-Practices – Configuración en control de versiones, builds/configuración reproducibles. – Estrategias de ramificación en control de versiones. Ramas de desarrollo vs ramas de lanzamiento. Desarrollo basado en tronco (*trunk-based development*). – Estrategias de fusión/rebase, cuando sean relevantes. • Gestión de lanzamientos (*release management*). • Automatización de procesos de software: – Sistemas de construcción (*build systems*): el valor de builds rápidos, herméticos y reproducibles, comparar/contrastar enfoques para construir un proyecto. – Integración Continua (CI): el uso de automatización y pruebas automatizadas para hacer una validación preliminar de que la revisión actual del tronco se construye y pasa las pruebas (básicas). – Despliegue Continuo (CD): el uso de automatización para liberar de forma automática cada cambio que supera las pruebas hacia el entorno de producción, garantizando entregas frecuentes y confiables. – Gestión de dependencias: actualización de dependencias externas/aguas arriba, gestión de paquetes, SemVer. • Herramientas de prueba, incluyendo herramientas de análisis estático y dinámico. Fundamentos de Desarrollo de Software (SDF)-Practices,SEC-Coding • Conceptos y mecanismos de integración de herramientas. Fundamentos de Desarrollo de Software (SDF)-Practices • Cultura DevOps: flujo, retroalimentación y aprendizaje continuo.	• Describir la diferencia entre la gestión de configuración de software centralizada y distribuida [Describir] • Describir cómo el control de versiones puede usarse para ayudar en la gestión de lanzamientos de software [Describir] • Comprender el uso de sistemas de CI/CD como una fuente de verdad para el estado del código compartido del equipo (éxito en la construcción y pruebas) [Explicar] • Demostrar la capacidad de usar herramientas de software para apoyar el desarrollo de un producto de software de tamaño mediano [Demostrar]

Lecturas : [Kim+21]

Unidad 7: Control de Versiones y CI/CD (6 horas)	
Resultados esperados: AG-C11,AG-C12	
Temas	**Aprendizaje esperado (*Learning Outcomes*)**
• Principios de IaC: reproducibilidad, idempotencia y versionamiento de infraestructura. • Enfoques declarativo vs. imperativo. • Terraform: proveedores, recursos, módulos y gestión de estado. • Herramientas nativas de nube: AWS CloudFormation, Azure ARM Templates, GCP Deployment Manager. • Gestión de configuración de software y control de versiones: Fundamentos de Desarrollo de Software (SDF)-Practices – Configuración en control de versiones, builds/configuración reproducibles. – Estrategias de ramificación en control de versiones. Ramas de desarrollo vs ramas de lanzamiento. Desarrollo basado en tronco (*trunk-based development*). – Estrategias de fusión/rebase, cuando sean relevantes.	• Escribir y aplicar configuraciones de Terraform para aprovisionar infraestructura en la nube. [Usar] • Identificar elementos de configuración y usar una herramienta de control de código fuente en un proyecto pequeño basado en equipo [Analizar] • Comparar herramientas de IaC según sus capacidades de gestión de estado y modularidad. [Evaluar]
Lecturas : [Mor21], [Kim+21]	

Unidad 8: Aplicaciones en Red (6 horas)	
Resultados esperados: AG-C09,AG-C11	
Temas	**Aprendizaje esperado (*Learning Outcomes*)**
• Modelos de servicio en la nube: IaaS, PaaS, SaaS, FaaS. • Paradigmas de aplicaciones distribuidas (por ejemplo, cliente/servidor, peer-to-peer, nube, edge y fog). Ver también: Computación Paralela y Distribuida (PDC)-Communication, Computación Paralela y Distribuida (PDC)-Coordination • Principales proveedores: AWS, Google Cloud, Azure — servicios clave. • Servicios de cómputo: máquinas virtuales, funciones serverless, contenedores administrados. • Servicios de almacenamiento: objetos, bloque y bases de datos administradas. • Redes en la nube: VPCs, balanceadores de carga, CDN.	• Analizar las necesidades de demandas específicas de aplicaciones en red [Analizar] • Comparar los modelos IaaS, PaaS, SaaS y FaaS para seleccionar el más adecuado a una arquitectura dada. [Evaluar] • Diseñar una aplicación nativa de la nube usando servicios administrados de un proveedor principal. [Usar]
Lecturas : [Kim+21], [Bur+22]	

Unidad 9: Ingeniería de Confiabilidad (6 horas)	
Resultados esperados: AG-C11,AG-C12	
Temas	**Aprendizaje esperado (*Learning Outcomes*)**
• Análisis basado en mediciones de la confiabilidad del software - telemetría, monitorización y alertas, paneles de control, métricas de calificación de lanzamientos, etc. • Los tres pilares de la observabilidad: métricas, logs y trazas distribuidas. • Trazas distribuidas e identificadores de correlación. • Prácticas de ingeniería de confiabilidad de software - incluyendo revisiones, pruebas, verificación de modelos práctica (practical model checking) • Estrategias de alertas y prácticas de guardia (*on-call*).	• Describir cómo determinar el nivel de confiabilidad requerido por un sistema de software [Describir] • Identificar métodos que conducirán a la realización de una arquitectura de software que logre un nivel especificado de confiabilidad [Analizar] • Implementar observabilidad en una aplicación de nube usando métricas, logs y trazas distribuidas. [Usar]
Lecturas : [Kim+21]	

Unidad 10: Confiabilidad del Sistema (6 horas)	
Resultados esperados: AG-C09,AG-C12	
Temas	**Aprendizaje esperado (*Learning Outcomes*)**
• Concepto de confiabilidad como probabilidad de fallo o tiempo medio entre fallos, y fallas (faults) como causa de fallos (failures) • Comparar el enfoque de "presupuesto de error" (error budget) para la confiabilidad con el enfoque "libre de errores" (error-free) e identificar dominios donde cada uno es relevante. • Técnicas y modelos de tolerancia a fallos de software – Diferencias contextuales en la tolerancia a fallos (por ejemplo, hacer fallar un sistema crítico para la aviación se evita firmemente, hacer fallar un sistema de procesamiento de datos antes de que se escriban datos corruptos en el almacenamiento es muy valioso) • Confiabilidad mediante redundancia: – verificación y reintento (*check and retry*) Sistemas Operativos (OS)-Faults,NC-Reliability – codificación redundante (códigos de corrección de errores, CRC, FEC, RAID) Arquitectura y Organización (AR)-MemoryHierarchy,NC-Reliability,DM-DistributedDatabasesCloudComputing – duplicación/reflejo (*mirroring*)/réplicas Gestión de Datos (DM)-DistributedDatabasesCloudComputing • Confiabilidad del software, confiabilidad del sistema y comportamiento de fallos • Indicadores de nivel de servicio (SLI), objetivos (SLO) y acuerdos (SLA).	• Identificar formas de aplicar redundancia para lograr tolerancia a fallos [Analizar] • Identificar dependencias de punto único de fallo (single-point-of-failure, SPF) en un diseño de sistema [Analizar] • Explicar los problemas que existen para lograr niveles muy altos de confiabilidad [Explicar] • Definir SLIs, SLOs y presupuestos de error para un servicio en producción. [Usar]
Lecturas : [Kim+21]	

8. PLAN DE TRABAJO

8.1 Metodología

Se fomenta la participación individual y en equipo para exponer sus ideas, motivándolos con puntos adicionales en las diferentes etapas de la evaluación del curso.

8.2 Sesiones Teóricas

Las sesiones de teoría se llevan a cabo en clases magistrales donde se realizarán actividades que propicien un aprendizaje activo, con dinámicas que permitan a los estudiantes interiorizar los conceptos.

8.3 Sesiones Prácticas

Las sesiones prácticas se llevan en clase donde se desarrollan una serie de ejercicios y/o conceptos prácticos mediante planteamiento de problemas, la resolución de problemas, ejercicios puntuales y/o en contextos aplicativos.

9. SISTEMA DE EVALUACIÓN

********* EVALUATION MISSING ********

10. BIBLIOGRAFÍA BÁSICA

[RF20] Mark Richards and Neal Ford. *Fundamentals of Software Architecture: An Engineering Approach*. 1st. O'Reilly Media, 2020.

[BCK21] Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice*. 4th. Addison-Wesley Professional, 2021.

[Kim+21] Gene Kim et al. *The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations*. 2nd. IT Revolution Press, 2021.

[Mor21] Kief Morris. *Infrastructure as Code: Dynamic Systems for the Cloud Age*. 2nd. O'Reilly Media, 2021.

[New21] Sam Newman. *Building Microservices: Designing Fine-Grained Systems*. 2nd. O'Reilly Media, 2021.

[Bur+22] Brendan Burns et al. *Kubernetes: Up and Running*. 3rd. O'Reilly Media, 2022.