Technische Berufsschule Zürich TBZ
Höhere Fachschule

Semesterarbeit Einreichungsformular / Klasse ITCNE24 – 4. Semesterarbeit

Im Folgenden beschreibt der Studierende das geplante Thema seiner Zertifikatsarbeit. Der Studierende orientiert sich dabei an dem Bewertungsraster. Der begleitende Dozent entscheidet, ob es sich dabei um ein geeignetes Thema handelt und fügt seine Kommentare und Überlegungen hinzu.

Bitte die *kursiven Textteile* durch die konkreten Angaben ersetzen.

Name und Vorname des Studierenden

Miguel Schneider

Bei einer Teamarbeit ist hinter dem Namen die Rolle/Funktion aufzuführen, z.B. Leitung, Mitarbeit, Infrastruktur etc,

Titel der Semesterarbeit

Migration und Deployment eines Lizenzüberwachungstools auf eine Cloud-Native-Core-Architektur mittels Kubernetes und DevOps

Zu behandelnde Themenfelder / Module (bitte ankreuzen)

	Pkt.	Themenfeld / Modul	
1.		Projektmanagement (Corrado)	Pflicht
2.		DevOps* (Armin, Thanam, Patrick)	Pflicht
3.		CNC - Cloud-native Core* (Philip S.)	Pflicht
		Themen aus dem 3. Semester	
4.		Relationale Datenbanken (Yves)	☐
5.		NoSQL, Key/Value, Time Series (Yves)	☐
6.		Microservices mit Python (Boris)	☐
7.		Datensicherheit / Datenschutz (vakant)	☐
8.		Netzwerk (Raphael)	☐

* Als Infrastruktur wird Kubernetes mit mindestens einer CI/CD Pipeline erwartet.

Kurzbeschreibung des Zertifikatsarbeit Themas (1 bis max. 2 Seiten)

Problemstellung / Ausgangslage / Potential der Semesterarbeit

Das bisher entwickelte Lizenzüberwachungstool für die ISE AG läuft in einer statischen Umgebung und erfordert manuelle Eingriffe bei der Bereitstellung, Skalierung und Wartung der Microservices. Diese Architektur ist nur begrenzt flexibel und erschwert eine schnelle Umsetzung von Änderungen oder Erweiterungen.

*Im Rahmen dieser Semesterarbeit soll das bestehende System in Richtung einer **Cloud-Native-Core-Architektur** weiterentwickelt werden. Ziel ist es, eine **automatisierte, skalierbare und wartungsfreundliche Bereitstellung** der Microservices zu ermöglichen.*

*Zu Beginn wird eine **Evaluation der Infrastruktur** durchgeführt, um zu entscheiden, ob die Umgebung lokal oder cloudbasiert betrieben wird. Diese Evaluationsphase ist bewusst Teil der Zielsetzung, um fundierte technische Entscheidungen zu treffen und die Lösung flexibel an die vorhandenen Rahmenbedingungen anzupassen.*

*Darauf aufbauend wird eine **CI/CD-Pipeline** konzipiert und umgesetzt, welche die Automatisierung zentraler Entwicklungs- und Deploymentprozesse ermöglicht. Die bestehende Flask-Service-Struktur aus der vorherigen Semesterarbeit dient als Grundlage und wird bei Bedarf um DevOps- und Container-Funktionalitäten erweitert.*

Die Arbeit verbindet damit praxisorientierte Cloud- und DevOps-Technologien und legt den Fokus auf methodisches Vorgehen, Nachvollziehbarkeit und eine schrittweise, begründete Weiterentwicklung der bestehenden Lösung.

- ***A**mbitioniert --> nicht zu einfach, nicht zu schwer*
- ***M**otivierend --> persönliches Interesse*
- ***O**rganisiert --> sind die Hilfsmittel vorhanden / organisierbar*
- ***R**ealistisch --> Zeitmanagement*
- ***E**cht --> Bezug zu den Vorgaben*

Zielsetzung der Semesterarbeit

- **Evaluation und Festlegung der Infrastruktur**
  Zu Beginn des Projekts wird eine Evaluation durchgeführt, um die geeignete Infrastruktur für den Betrieb der Anwendung zu bestimmen, entweder lokal oder in einer Cloud-Umgebung. Dabei werden technische Anforderungen, Sicherheitsaspekte und vorhandene Ressourcen berücksichtigt. Das Ergebnis dient als Grundlage für die weitere Umsetzung, bleibt jedoch offen für Anpassungen bei neuen Erkenntnissen im Projektverlauf.

- **Aufbau einer CI/CD-Pipeline zur Automatisierung zentraler Prozesse**
  Es wird eine CI/CD-Pipeline entwickelt, die den automatisierten Ablauf von Build-, Test- und Deployment-Prozessen ermöglicht. Ziel ist eine durchgängige Automatisierung und Nachvollziehbarkeit der Entwicklungsabläufe. Die konkrete Toolwahl (z. B. GitHub Actions oder GitLab CI) und der Umfang einzelner Pipeline-Stufen können im Projektverlauf iterativ angepasst werden.

- **Weiterentwicklung in Richtung einer Cloud-Native-Core-Architektur**
  Das bestehende Lizenzüberwachungstool wird so erweitert, dass zentrale Cloud-Native-Prinzipien umgesetzt werden – insbesondere Containerisierung, Skalierbarkeit und modulare Strukturen. Dabei wird flexibel vorgegangen: Umfang und Tiefe der Umsetzung können auf Basis der Ergebnisse aus der Evaluationsphase angepasst werden, um die technische Machbarkeit und den Nutzen sicherzustellen.

- **Sicherstellung von Datenschutz, Stabilität und Betriebssicherheit**
  Während der Entwicklung werden Datenschutz- und Sicherheitsaspekte fortlaufend berücksichtigt. Zudem sollen geeignete Mechanismen zur Überwachung, Fehlertoleranz und Stabilität implementiert oder getestet werden. Welche konkreten Lösungen (z. B. Monitoring-Tools, Self-Healing, Logging) eingesetzt werden, wird im Verlauf evaluiert und dokumentiert.

Terminplan mit den wesentlichen Arbeitsschritten

Datum	Aktivität	Wer	Empfänger
01.10.25	Ablauf Semesterarbeiten und Liste mit Projektthemen vorstellen	Lehrgangsleitung	Studierende
20.10.25	Abgabe Einreichungsformular	Studierende	Experte/innen
27.10.25	Freigabe Semesterarbeit durch Expert/innen	Experte/innen	Studierende
27.10.25	Beginn Umsetzung Semesterarbeit	Studierende	Experte/innen
17.11.25	1. Sprint, evtl. mit Besprechung	Studierende	Experte/innen
15.12.25	2. Sprint, evtl. mit Besprechung	Studierende	Experte/innen
23.01.26	3. Sprint, evtl. mit Besprechung	Studierende	Experte/innen
28.01.26	Abgabe der Arbeit / Abnahme mit Schlusspräsentation	Studierende	Experte/innen
04.02.26	Notenvorschlag	Projekt-Experte/in	Lehrgangsleitung
13.02.26	Mitteilung der Noten mit individuellem Feedback zur Einsicht hochladen	Projekt-Experte/in	Studierende
13.03.26	Mitteilung der Noten	Lehrgangsleitung / Sekretariat	Studierende

Sachmittel / Rahmenbedingungen

Für die erfolgreiche Umsetzung der Semesterarbeit und zur Erstellung der Projektergebnisse werden folgende Sachmittel und Rahmenbedingungen benötigt:

1. **Kubernetes-Umgebung**
   Zur Orchestrierung und Verwaltung der containerisierten Microservices. Die Umgebung dient als Laufzeitplattform für die Bereitstellung und Skalierung des Lizenzüberwachungstools.

2. **Docker Container (Docker Desktop)**
   Zur Containerisierung und Entwicklung der bestehenden Flask-Microservices. Docker stellt die Basis für den späteren Betrieb innerhalb von Kubernetes dar.

3. **CI/CD-Pipeline (GitLab oder GitHub Actions)**
   Für die automatisierte Erstellung, das Testen und das Deployment der Anwendung. Änderungen am Quellcode werden kontinuierlich integriert und ausgeliefert.

4. **Bestehende Flask-Service-Struktur (aus vorheriger Semesterarbeit)**
   Als technische Grundlage dient die im letzten Semester entwickelte Flask-Anwendung, die nun um DevOps- und Cloud-Native-Funktionalitäten erweitert wird.

5. **Microsoft 365 Test-Tenant (ISE AG)**
   Zur Abfrage und Verarbeitung von Lizenzinformationen über die Microsoft Graph API. Dieser Tenant stellt die erforderlichen Zugriffsrechte und Testdaten bereit.

6. **GitLab / GitHub Repository**
   Für die Versionsverwaltung, Aufgabenorganisation (Scrumban) und Projektdokumentation.

Vorgaben, Methoden und Werkzeuge

Im Rahmen dieses Projekts werden definierte Methoden, Werkzeuge und technische Vorgaben eingesetzt, um eine strukturierte, praxisnahe und zielgerichtete Umsetzung sicherzustellen.

Methoden:

- **DevOps-Prinzipien –** Kombination von Entwicklung und Betrieb zur durchgängigen Automatisierung von Build-, Test- und Deployment-Prozessen.
- **Scrumban –** Agile Aufgabenplanung mit visueller Darstellung, laufender Priorisierung und flexibler Sprintplanung.
- **Lean Development –** Fokus auf effiziente, ressourcenschonende Abläufe und kontinuierliche Verbesserung.
- **Infrastructure as Code (IaC) –** Definition und Verwaltung der Infrastruktur (z. B. Kubernetes-Ressourcen) über deklarative Konfigurationen.

Werkzeuge:

- **GitLab / GitHub Actions –** Aufbau und Betrieb einer CI/CD-Pipeline zur automatisierten Bereitstellung der Microservices.
- **Kubernetes (lokal oder Cloud) –** Orchestrierung und Verwaltung der containerisierten Microservices.
- **Docker –** Containerisierung der bestehenden Flask-Services.
- **Helm –** Paketmanagement für Kubernetes-Deployments.
- **Visual Studio Code –** Hauptentwicklungsumgebung für Backend, Frontend und Pipeline-Skripte.
- **Azure CLI / kubectl –** Verwaltung und Überwachung der Kubernetes-Cluster.
- **Python / Flask –** Framework für die bestehenden Microservices.
- **Microsoft 365 Graph API –** Lizenzabfragen aus den vorhandenen Test-Tenants.
- **Git –** Versionsverwaltung und Nachverfolgung der Code-Änderungen.

Vorgaben:

- **Verwendung des bestehenden Lizenzüberwachungstools** aus der vorherigen Semesterarbeit als Grundlage.
- **Deployment in einer Kubernetes-Umgebung** mit lokalem oder Cloud-basiertem Cluster.
- **Implementierung einer CI/CD-Pipeline** als Pflichtbestandteil gemäss den DevOps-Vorgaben.
- **Einhaltung von Datenschutzrichtlinien**, gemäss interner Richtlinien der ISE AG und geltenden Datenschutzvorgaben.
- **Dokumentation und Fortschrittsverfolgung** erfolgen über GitHub

Risiken

Risiko	Eintritt	Auswirkung	Massnahme zur Vermeidung / Minderung
Komplexität bei der Einrichtung der CI/CD-Pipeline (z. B. YAML-Fehler, Pipeline-Trigger)	**Mittel**	Mittel	Schrittweise Implementierung, Nutzung von Templates, regelmässige Tests und Code-Reviews
Fehlkonfiguration von Kubernetes-Ressourcen (z. B. Pods, Services, Secrets)	**Mittel**	Hoch	Einsatz von Helm-Charts, Validierung der Konfiguration über Testumgebungen und Dokumentation der Deployments
Datenschutzrisiko durch unsichere Speicherung von Lizenzdaten	**Niedrig**	Hoch	Klare Trennung von sensiblen Daten, Speicherung innerhalb der definierten Systemumgebung und Einhaltung interner Datenschutzrichtlinien
Integrationsprobleme zwischen Microservices oder Pipeline-Komponenten	**Mittel**	Mittel	Klare Schnittstellendefinitionen, Integrationstests in jeder Pipeline-Stufe, Logging und Fehlerauswertung
Instabilität des Systems bei Skalierung oder Lasttests	**Mittel**	Hoch	Stufenweiser Ausbau der Skalierung, Tests unter realistischen Bedingungen, Nutzung von Kubernetes-Self-Healing-Mechanismen
GitHub-Dokumentation wird nicht laufend gepflegt	**Niedrig**	Niedrig	Doku fix in Workflow einplanen, regelmässige Erinnerung im Taskboard
Authentifizierungsprobleme mit der Microsoft Graph API in Kubernetes (z. B. Zertifikate, Thumbprints, Secrets)	**Mittel**	Hoch	Sichere Verwaltung und regelmässige Erneuerung der Zertifikate/Secrets, Einsatz von Kubernetes-Secrets oder Key Vault, Monitoring und Logging der Authentifizierung.

Entscheid des begleitenden Dozenten

Bitte ankreuzen

- Genehmigt ☐
- Zu verbessern ☐
- Abgelehnt ☐

Begleitender Dozent

Ort und Datum: _____

Name & Unterschrift: _____

Beurteilung des Antrages

Kriterium	Kommentare	Erfüllt
Anforderungen an die Form (Strukturierung) der Semesterarbeit		
Problemstellung		☐
Ziele (mindestens drei!)		☐
Risiken bezogen auf Zertifikatsarbeit		☐
Themenfelder (mindestens zwei!)		☐
Anforderungen an Qualität der Semesterarbeit		
Machbar		☐
Praxisnah		☐
Herausfordernd		☐
Lehrgangsbezug		☐

Damit die Semesterarbeit angenommen wird, müssen alle Kriterien erfüllt sein.

Die Kommunikation wird über den jeweiligen Teams-Kanal geführt.