Continuous Build and Delivery Assignment 2025 – Weighting 30%

a. Assignment Outline

The objective of this assignment is to take one microservice (it can be one of the microservices you developed in the MicroServices module) and follow it though a complete CI/CD pipeline that you have configured using appropriate tools. The objective is to automate as much of the flow from code check in by developer to application deployment. The pipeline should have a high level of test automation and code analysis. You can use local machine or AWS or a combination of both to deploy and demonstrate the pipeline. One option is to use localhost for the CI and AWS for CD.

b. What to include

Version Control

a. Use version control on developer machine

b. Push code to a remote repository. (Bitbucket or Github)

Build

c. Clone from the remote repository and build the application using Maven (or equivalent. build tool). Code should be packaged into a deployable entity such as a .war file.

Code Analysis

d. Set up a continuous code quality analysis server such as Sonarqube (recommended) and/or use PMD or Checkstyle to analyse the code.

e. Use Jenkins (CI server) to trigger this process on code push

Testing

f. Develop a test strategy (referring to the Test Pyramid), utilising the different test levels and test tools in your approach to achieve a high level of test automation.

g. Use Jenkins (CI server) to trigger this process on code push

Deployment

h. Create a docker image.

i. Set up e.g. ansible with ansible playbook to deploy docker container.

j. Use Jenkins or equivalent to automate the process

k. Deploy docker container on an application server

c. Submission

Submit a **word** document that

- Provides an **introduction** to CI/CD, explaining the key differences between CI and CD. Write one to two pages on current state of the art practices/tools in this space. Reference any resources that you use.
- List the **User Stories** for the Microservice (in Who/What/Why format with Acceptance Criteria) that you developed.
- Outline the **application architecture** on a high level
- Document your **test strategy**, how does your strategy align with the Test Pyramid.
- Fully describe your **pipeline and all its stages**. (Three to four sentences on each stage or part of the process. List the key points/commands for configuring that stage). Use diagrams where appropriate. List the tools that you used in each stage with reasons why. (reference resources). If you have **code analysis**, show the output report and show the **test results** and **test coverage**. Describe the test that you ran and provide the code or code snippets.
- **Evaluate** your pipeline including the test strategy. How does it perform (time)? How much of the flow is automated.? Would you select an alternative tool to the ones you have chosen?

Submit a **screencast** with a demo of your pipeline.

- Give a brief context and rational for your pipeline
- Show the pipeline executing. If the complete pipeline has not been automated, show the manual steps. If the pipeline is fully automated, go back and show the outcome of each stage. (E.g. test results, code analysis report, jar/war file deployed).
- The demonstration should what happens when a small code change is pushed to the remote repository and how the pipeline executes from there.
- There is no minimum time for the screencast but it should **not** be longer than 8 minutes.

d. Marking Rubric

Elements	Excellent (70+)	Good (55%-69%)	Satisfactory (40%-55%)	Fail (0-39%)
Presentation (10%)	Clear, well-paced narration. Engaging and structured presentation.. Effective use of diagrams, workflow charts, and demos	Mostly clear but lacks minor structuring. Some use of diagrams and visuals.	Some issues with clarity or flow.. Minimal visuals.	Difficult to understand. Unstructured presentation. No supporting visuals..
Context and Rationale (10%)	Clear, well-structured explanation of the CI/CD pipeline. Strong justification for tool choices. Insightful discussion on DevOps, modern trends, and best practices	Good explanation with some justification for tool selection Some references to best practices.	Basic explanation with limited justification. Minimal reference to industry trends	No rationale or justification provided. No mention of DevOps principles or best practices.
Pipeline Demonstration (40%)	Strong test automation strategy with multiple levels (unit, integration, end-to-end). Alignment with test pyramid. Quality gates implemented in sonar. Minimal technical debt. Deployment to cloud (AWS, Ansible, Kubernetes, or alternative)..	Partial automation with some manual steps. Good test automation but not aligned to test pyramid. Local deployment considered.	Mostly manual pipeline with basic automation. Basic test strategy with minimal automation. Basic deployment without automation.	No working pipeline No meaningful test automation. No deployment considered.

Screencast (60%)

Elements	Excellent (70+)	Good (55%-69%)	Satisfactory (40%-55%)	Fail (0-39%)
Organisation and Presentation (10%)	Logical, well-organized, and professional.	Mostly clear and structured.	Somewhat structured but lacks consistency	Poorly written, lacks logical structure. No referencing Havard- (may be a breach of academic integrity)
Content (Introduction, User Story, Architecture and Pipeline) 10%	Clear breakdown of pipeline with diagrams and explanations..	Good breakdown but lacks detail in some areas.	Basic explanation with minimal diagrams.	Pipeline explanation is unclear or missing .
Testing and Code Quality 10%	Comprehensive strategy using Test Pyramid. Includes Unit, Integration, and End-to-End testing. Static analysis (SonarQube, Checkstyle, or PMD) and security scanning (e.g. OWASP considered). 80%+ coverage, use of mocks and automated reporting	Good test coverage but missing some test levels – lack of alignment to test pyramid Some code quality tools used, minimal security scanning, no quality gates. 60-79% coverage, some manual testing.	Limited test automation (e.g., only unit tests). Basic tests with low automation.	No test strategy or automation. No security or quality checks. No meaningful test coverage.
Evaluation 10%	Evaluates build time, test efficiency, and performance bottlenecks. . Identifies weaknesses in the pipeline and suggests improvements..	Some performance analysis but lacks depth. Identifies some weaknesses but lacks alternatives	Minimal performance evaluation. Few observations, lacks depth or focus	No evaluation provided. No critical evaluation.

Report (40%)

CheckList:

- Upload to moodle by Sunday 7th April. 2024 17.00. .zip with your code, screencast and report.
- Live Demo/Q&A via zoom: You will be notified via student email if you need to participate in a Live Q&A.