Capstone Grand Challenge: Evolving an Open-Source Orchestrator from TOSCA 1.3 to TOSCA 2.0

# Context

Cloud-native orchestration is governed by standards such as OASIS TOSCA (Topology and Orchestration Specification for Cloud Applications). Existing open-source engines like Puccini (a TOSCA compiler) and Khutulun (an orchestrator) are stable for TOSCA 1.3, but the new TOSCA 2.0 specification radically changes assumptions:  
- Profiles are externalized and must be imported by name.  
- Operational models formalize in-life changes (updates, upgrades, reactive operations).  
- The CSAR packaging format is updated.  
- Path traversal and function semantics are expanded, including user-defined functions.  
  
Migrating a production-grade orchestrator to support this new version is a real-world example of software evolution under changing standards, requiring architectural foresight, rigorous testing, and attention to quality attributes (e.g., portability, maintainability, backward compatibility).

# The Challenge

Your task is to design, implement, and validate a proof-of-concept evolution of Puccini + Khutulun from TOSCA 1.3 to 2.0.  
  
This is not merely a code-migration exercise: it is an open-ended evolution challenge that touches on requirements engineering, software architecture, quality assurance, and DevOps practices.

# Learning Objectives

1. Software Evolution: Experience requirements drift due to evolving standards; assess technical debt and architectural fitness for change.  
2. Software Quality: Apply ISO/IEC 25010 quality attributes (maintainability, portability, interoperability); design regression/conformance testing.  
3. Open-Source Engineering: Work with real-world projects, learn contribution workflows, documentation, and community interaction.  
4. Research Skills: Critically analyze a formal standard (TOSCA 2.0) and translate abstract specifications into concrete design and implementation.

# Expected Deliverables

1. Evolution Plan: Analysis of TOSCA 1.3 vs 2.0; roadmap for parser, orchestrator, and CSAR adaptations.  
2. Implementation Prototype: Extend Puccini to parse and compile TOSCA 2.0; extend Khutulun for operational model; define and publish a minimal Khutulun Profile.  
3. Quality Assurance Artifacts: Test suite with community and custom CSARs; backward compatibility evaluation; CI pipeline.  
4. Capstone Report & Demo: Documentation (30–40 pages) with architecture decisions, challenges, and trade-offs; live demo of PoC CSAR deployment.

# Software Architecture (High-Level)A diagram of a software company AI-generated content may be incorrect.

The evolved orchestrator consists of the following components:  
- \*\*TOSCA Parser (Puccini extension):\*\* Handles TOSCA 2.0 grammar, profiles, and CSAR v2.0.  
- \*\*Intermediate Representation (Clout):\*\* Extended to include operational model hooks and expanded function semantics.  
- \*\*Orchestrator Engine (Khutulun):\*\* Interprets Clout, executes workflows and policies, and manages in-life changes (scale, update, upgrade).  
- \*\*Profile Repository:\*\* Contains the minimal Khutulun Profile (v0.1.0) defining compute, container, network, lifecycle, and policies.  
- \*\*Execution Adapters:\*\* Implement lifecycle operations (create, configure, start, stop, scale, update) against Docker/Kubernetes or a mock runtime.  
- \*\*CI/CD Pipeline:\*\* Automated validation of conformance with TOSCA 2.0 test suites and backward compatibility with TOSCA 1.3 templates.

# Hints for Technical Specifications

- Update Puccini to recognize 'tosca\_definitions\_version: 2.0' and handle profile imports by name.  
- Extend expression evaluator for TOSCA Path, new functions, and user-defined functions.  
- Implement CSAR v2.0 metadata format in 'TOSCA.meta'.  
- Add operational model deltas (scale, update, rolling upgrade) in Clout and orchestrator workflows.  
- Publish a minimal profile 'khutulun.profile@0.1.0' with ~10 node, capability, and policy types.  
- Provide a PoC CSAR ('hello-web') that deploys a containerized service, scales it, and performs a rolling upgrade.

# Assessment Rubric

- Technical Depth (40%): Successful parser/orchestrator evolution and correctness of 2.0 features.  
- Software Quality Practices (25%): Testing, CI/CD, maintainability analysis, and documentation.  
- Research/Analysis (20%): Depth of gap analysis and mapping of spec clauses to design.  
- Presentation/Demo (15%): Clarity of report, quality of live demo, and reproducibility.