CGE-P Capstone Project Overview

GRC Engineering Club Certification

Plan your capstone project as you go through the course.

Everything you learn in CGE-P — Infrastructure-as-Code, Policy-as-Code, CI/CD compliance pipelines, evidence management, and OSCAL — comes together in one GitHub repository you design and build yourself. This document gives you the full picture on day one: the scenario, the four layers you'll build, what gets graded and how, and a week-by-week plan to get there. Keep it open as you go through the course.

Credential	CGE-P (Certified GRC Engineer — Practitioner)
Pass Threshold	Weighted average ≥ 65 across 8 dimensions, no auto-fail triggers
Deliverables	Public GitHub repo · Signed evidence bundle · WRITEUP.md
Time Estimate	30 days (1 wk design · 1 wk infra · 1 wk policy+pipeline · 1 wk OSCAL+write-up)
Starter Repo	github.com/GRCEngClub/cgep-app-starter (fork it first)

1 · The Scenario

You are the **first GRC engineer at Acme Health**, a 50-person telehealth company. The engineering team has shipped a Patient Intake API. It works. It is not audit-defensible. The CTO has asked you to make it audit-defensible in **30 days** — without slowing the engineering team down.

The starter is real code in a real repo: **GRCEngClub/cgep-app-starter**. Fork it, deploy it with *make deploy*, confirm *make test* returns a received submission, then govern it. Inheriting a working system and making it defensible — that's the whole brief.

Your Framework Choice

Acme is pursuing three flags simultaneously. You pick **one as your primary framework**, defend the choice in your write-up, and structure every layer around it. See FRAMEWORKS.md in the starter for full primers.

Framework	Why Acme Is Considering It
HIPAA Security Rule	PHI is at stake — the Patient Intake API handles protected health information. Maps to Safeguards Rule technical controls.
SOC 2 Type II	An enterprise customer is asking. Maps to Trust Services Criteria (Security, Availability, Confidentiality). Requires evidence of continuous control operation.
CMMC Level 2	A federal pilot is on the table. Maps to NIST 800-171 rev 3 / CMMC L2 practices. Requires a full 110-practice control implementation and evidence trail.

The starter ships with **eight named, intentional gaps** (GAPS.md). Your job is to close them and produce evidence that the system stays closed.

2 · What You'll Build — Four Layers, One Repo

Each layer maps to a course chapter. The repo proves you can integrate all four.

Layer 1 · Terraform Baseline (terraform/)

- KMS key(s) with rotation enabled — bring the starter's S3 + DynamoDB under your CMK
- S3 evidence bucket with Object Lock (COMPLIANCE or GOVERNANCE mode) — versioned, KMS-encrypted
- CloudTrail trail (multi-region, log-file-validation on) writing to a dedicated bucket
- Gap-closing hardening overrides on the starter's resources (e.g., KMS SSE on S3, Lambda VPC config, IAM tightened from dynamodb:*)
- Use the starter's VPC — don't build a second one

Layer 2 · OPA Policy Suite (policies/)

- 5+ Rego policies enforcing controls from your declared primary framework
- Each policy: metadata block (framework, control ID, severity, remediation)
- Each policy: its own _test.rego with passing and failing fixtures
- Each policy: catches a real gap from GAPS.md — not a generic tag check
- Conftest runs the suite against your Terraform plan in the pipeline; the pipeline fails closed

Layer 3 · GitHub Actions Pipeline (.github/workflows/grc-gate.yml)

- Step 1 — Plan the Terraform
- Step 2 — Policy check with Conftest
- Step 3 — Apply on merge to main
- Step 4 — Sign the evidence bundle with Cosign (keyless, via GitHub OIDC)
- Step 5 — Upload the signed bundle to the Layer 1 evidence vault
- Repo history must show one green PR (merged) and one red PR (blocked by policy gate)

Layer 4 · OSCAL Component (oscal/components/)

- One component-definition.json describing what you actually built
- Real UUIDs — not placeholders
- source field on each control-implementation points at your declared framework's catalog
- Implementation statements reference real Terraform resources (addresses or ARNs as props)
- Evidence links resolve to real signed objects in your vault
- A profile selecting the controls your component implements

3 · Three Deliverables

Submit the repo URL plus the commit SHA you want graded. The write-up is not optional.

Artifact	Format	What It Proves
Public GitHub repo	Terraform + Rego + YAML	You can build the whole pipeline end-to-end.
Evidence bundle	Signed .tar.gz in S3 Object Lock vault	Your pipeline produces audit-grade, tamper-evident evidence.
WRITEUP.md	Markdown file in the repo	You can explain design choices, control coverage, trade-offs, and honest gaps to a stakeholder.

Three things we score hard: (1) End-to-end integration — the gate runs, decides, apply triggers signing, signing uploads. Not four disconnected demos. (2) Working evidence pipeline — Cosign signature verifies against Sigstore, SHA-256 recomputes, Object Lock retention holds. All three must pass. (3) Clear design reasoning — your write-up explains *why*, not just *what*.

4 · The Rubric — How You're Scored

Submissions are evaluated across **eight dimensions**. Each dimension has two tiers: **Tier 0** (deterministic static analysis — terraform fmt, checkov, gitleaks, conftest) and **Tier 1** (qualitative judgment against the rubric criteria below). **Pass = weighted average** ≥ **65** AND no auto-fail triggers fired.

Dimension	Wt.	What It Measures	90–100	70–89	50–69	0–49
Infrastructure-as-Code Quality	15 %	Compliance IaC is modular, validated, follows best practices	All Tier 0 pass; clean modules & state	Tier 0 mostly passes; minor boundary violations	Multiple Tier 0 failures or HIGH severity findings	No IaC or fails to validate
Policy-as-Code	15 %	Compliance requirements expressed as executable policies	Policies map to controls; comprehensive tests; deny-by-default	Policies tested; mapping partial	Policies present but no tests or no control mapping	No policy-as-code
CI/CD Compliance Integration	10 %	Compliance checks run automatically on every change	CI blocks non-compliant code; artifacts preserved; staged pipeline	CI runs scanners but informational-only or only on main	CI exists but doesn't run compliance scanners	No CI or purely cosmetic
Continuous Monitoring & Detection	10 %	System detects drift or misconfiguration in real time	Targeted detections mapped to controls; tested; alert routing defined	Detections present; partial coverage	Detection logic exists but generic or untested	No monitoring or detection logic
Evidence Automation & OSCAL	10 %	Compliance evidence generated automatically from system state	Wired collection; OSCAL or equivalent; provenance preserved	Collection scripts exist but not scheduled; OSCAL partial	Manual scripts only; no schema or scheduling	No evidence automation
Control Test Coverage	10 %	Controls are tested for effectiveness — not just declared	Comprehensive positive/negative tests; integration-level; tied to controls	Tests present and pass; coverage is partial	Token tests; no negative cases; not tied to controls	No tests or tests don't run
Control-to-Code Documentation	15 %	Auditor can trace any requirement to the specific code that satisfies it	Bidirectional mapping; standard control IDs; current; explains rationale	Mapping present but one-directional or partial	Some control refs but no real mapping; or invented IDs	No control-to-code documentation
Engineering Hygiene	15 %	Repository reflects professional software engineering practice	Clean structure; pinned deps; meaningful commits; no secrets/PII	Mostly clean; minor lints; commits adequate	Disorganized but functional; some HIGH semgrep findings	Secrets/PII present (auto-fail trigger)

5 · Auto-Fail Triggers

Any one of these causes an immediate fail regardless of your weighted score. Check these before you submit.

- ✗ Repository is private or inaccessible at time of grading
- ✗ Active secrets detected in the repository (verified non-test credentials)
- ✗ Repository contains real PII or production credentials
- ✗ Repository is a fork of a published reference solution with no meaningful modification
- ✗ Repository has no README

6 · Suggested 30-Day Plan

Break it into weeks. Don't start day 29. If you did the labs as you went, this is one week of design, two weeks of wiring, one week of writing.

Week	Focus
Week 1 · Design	Pick your framework. Map controls to gaps. Sketch the repo structure. Open a one-page design doc in the repo — it becomes the spine of WRITEUP.md.
Week 2 · Build Infra	Terraform baseline: evidence bucket with Object Lock, KMS key, CloudTrail, gap-closing overrides. Apply once by hand from a feature branch. Don't start the pipeline until the baseline applies clean.
Week 3 · Policy + Pipeline	Write 5+ Rego policies with tests. Build the GitHub Actions workflow. Wire Cosign signing. Open the green PR. Open a second PR that intentionally violates a policy and watch it go red.
Week 4 · OSCAL + Write-up	Author component-definition.json. Validate with trestle. Wire evidence URIs to real vault objects. Write WRITEUP.md (design decisions, control coverage, trade-offs, honest gaps). Submit repo URL + commit SHA.

If you're in week three with no baseline running, cut scope. Trade workload resources for a working pipeline. Every time. Small and integrated beats large and disconnected.

7 · Lab → Capstone Mapping

Labs are optional but strongly encouraged. If you did the labs, the capstone is an assembly exercise — you already built most of the pieces.

Lab	What It Gives You for the Capstone
Lab 2.3 — First Compliant Resource	S3 hardening pattern (encryption, public-access-block, versioning, tags). Drop into the gap-closing overrides on the starter's uploads bucket.
Lab 2.4 — Modules for Compliance	Reusable-module discipline. Wrap your KMS + S3 hardening as a module so dev and prod consume the same compliance floor.
Lab 2.5 — IaC as Compliance Evidence	Your evidence vault with Object Lock + capture-evidence.sh. The capstone vault IS Lab 2.5's vault.
Lab 3.3 — Writing Rego	Starting policy library: metadata block format, test fixture pattern, three policies you can adapt directly.
Lab 3.4 — Conftest + Terraform	scripts/policy-gate.sh. The capstone pipeline calls this directly.
Lab 4.3 — GRC Evidence Pipeline	.github/workflows/grc-gate.yml. The capstone pipeline IS this workflow with two extra steps.
Lab 4.4 — Chain of Custody	Cosign + S3 Object Lock signing pattern. Wires Lab 4.3 into Lab 2.5.
Lab 5.2 — AWS Security Services	CloudTrail + Config baseline. The capstone Layer 1 native-controls layer.
Lab 6.1 — Introduction to OSCAL	Component-definition skeleton. Replace Lab 6.1 controls with your declared-framework controls and re-validate.

8 · Submission Checklist

When all boxes are checked, send the repo URL and commit SHA to the submission form at cert.grcengclub.com.

☐	Your repo is a fork (or clear derivative) of cgep-app-starter. The starter's resources are still present and runnable.
☐	Your declared primary framework is named in WRITEUP.md's first paragraph and in your OSCAL component's control-implementation.source.
☐	terraform/ adds KMS keys, S3 evidence bucket with Object Lock, CloudTrail, and the gap-closing overrides.
☐	policies/ has 5 or more Rego policies with tests. opa test ./policies passes. Each policy cites a control ID from your declared framework.
☐	.github/workflows/grc-gate.yml runs Plan → Policy check → Apply → Sign → Upload.
☐	One green PR and one red PR are visible in repo history.
☐	At least one signed evidence bundle in your vault. Cosign verifies. SHA matches. Object Lock retention active.
☐	oscal/components/.json validates with trestle.
☐	WRITEUP.md covers framework choice, gap remediation, design trade-offs, and what you didn't get to.
☐	README.md is short, with verification instructions for the grader.

Ship the repo. Earn the cert.