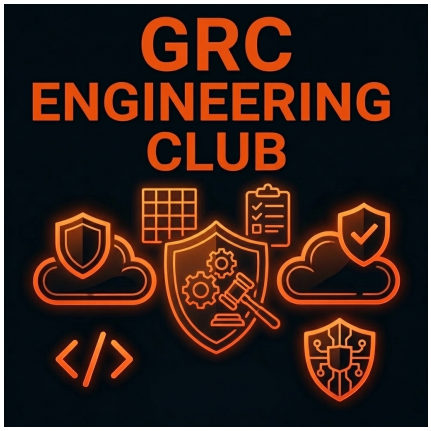


CGE-GCP

Study Guide

Certified GRC Engineer, GCP Specialty

GRC Engineering Club Training Academy

Instructor: AJ Yawn

Version 1.0 | November 2026

www.grcengclub.com

HOW TO USE THIS STUDY GUIDE

Overview of the Certification

The Certified GRC Engineer, GCP Specialty (CGE-GCP) is a cloud specialty stacked on CGE-P. It takes an engineer who can already write Terraform, encode policy, and defend evidence to an auditor, and teaches them to do it on Google Cloud by building one thing: a complete, automated GRC engineering pipeline in their own project. The pipeline discovers what is running, activates what is missing, stores evidence in a store you own, generates auditor deliverables, narrates them weekly, and remediates non-compliant resources through a least-privilege identity. The one-liner: don't map controls to Google Cloud. Build the pipeline that proves them.

Who Should Take This Exam

GRC engineers, compliance engineers, security engineers, and cloud-savvy auditors who hold CGE-P and work in, or are moving into, Google Cloud environments. Google Cloud makes choices that matter for compliance: the project is the unit of billing, APIs, IAM, and logs; workloads run as attached service accounts instead of keys; and every resource and IAM binding can be read from one inventory API. This course teaches that vocabulary from scratch, then assembles the services into an evidence pipeline.

Prerequisites

CGE-P is the only prerequisite, enforced in the platform. The course assumes you can read and write Terraform, use Git and pull requests, and explain a control objective, because CGE-P taught all three. No Google Cloud certification and no other CGE specialty is required.

Course Format and Study Time

About an hour of video: 25 short lessons plus 6 lab walkthroughs, with most lessons running 2 to 3 minutes. Two text articles (00_02 How to Use This Course, 07_04 Certification Exam Guide) and six hands-on labs complete the course. Total learner study time is 10 to 14 hours, and most of that is your hands on the keyboard: the labs, the capstone, and the exam.

Before Lab 1: Setup

The whole course runs in one Google Cloud project with no Organization above it, which is what a personal @gmail.com account gives you. Accounts on a company or custom domain put every new project under that domain's Organization, where your employer's policies apply to your lab. Create one project for the whole course, set a $25 budget with alerts at 50%, 90%, and 100% before anything else (a budget notifies, it does not stop spending), run both logins (gcloud auth login, then gcloud auth application-default login), and pin the project with gcloud config set project. Check gcloud config list before every lab. The article 00_02 and docs/SETUP.md in the repo cover each step.

The Six Labs

Every lab runs in your own project from your fork of the starter repo, github.com/GRCEngClub/cgegcp. One repo holds every lab and the capstone. Labs 3 to 6 build on each other, so full teardown happens once, at the end of Lab 6. The Google Cloud free trial gives new customers $300 in credit for 90 days with no automatic charge. Start it when you are ready to work, so the 90-day window covers Labs 1 through 6 and the capstone. Times and costs below come from the lab guides.

Lab	Name	Hands-on Time	Cost
1	Stand Up Your GCP GRC Sandbox	~30 min	$0
2	Turn It All On	~30 min	$0
3	Deploy Your Foundation	~45 min	$0; APIs and service accounts are free, the state bucket holds kilobytes
4	Evidence Flowing End to End	~60 min	Under $1; Firestore, Cloud Run functions, and Scheduler round to cents at lab volume
5	Generate Your First ATO Deliverables	~40 min	Under $1; the optional Gemini digest costs a fraction of a cent per run
6	Close the Loop	~60 min	Under $1

Recommended Study Timeline

Timeline	Activities	Estimated Hours
Week 1	Course introduction, Domains 1 and 2 (GCP Foundations, The GCP GRC Toolkit) + Labs 1 and 2. Account, budget, project, toolkit on.	2 to 3 hours
Week 2	Domains 3 and 4 (Terraform on GCP, The Evidence Plane) + Labs 3 and 4. Foundation deployed, evidence flowing daily.	3 to 4 hours
Week 3	Domains 5 and 6 (Reporting & Narrative, Enforcement & Operations) + Labs 5 and 6. Reports generating, loop closed.	2 to 3 hours
Week 4	Domain 7, the exam, then the capstone, while your scheduled collector builds run history.	3 to 4 hours

How to Use This Guide with the Video Course

This study guide complements the CGE-GCP video course, taught end to end by AJ Yawn. Each domain chapter below follows the lessons in order and condenses what each one teaches into prep notes. Watch the lesson, do the lab, then use these notes and the practice prompts to check yourself. Domains 1 through 3 get you fluent; Domains 4 through 6 build the pipeline stage by stage; Domain 7 is the capstone.

EXAM OVERVIEW

Exam Format

Attribute	Details
Total Questions	50 questions
Question Types	40 Multiple Choice + 10 Scenario-Based (built on real artifacts)
Time Limit	75 minutes
Pass Score	70% (35/50 correct answers)
Capstone	REQUIRED: the pipeline, deployed as code, graded pass/fail against a published rubric
Retake Policy	14-day wait between attempts; max 3 attempts per 12 months
Format	Computer-based (online proctored)
Price	$350, or free for active Club members

Domain Breakdown and Question Distribution

Domain	Topic	Weight	Questions	Study Focus
1	GCP Foundations	15%	8	Hierarchy, IAM bindings, role kinds, service accounts, keyless identity, shared responsibility
2	The GCP GRC Toolkit	15%	7	Asset Inventory, checks as code, control types, Cloud Audit Logs, the eight pipeline services
3	Terraform on GCP	15%	8	google provider, keyless auth, APIs as code, state, additive IAM, the policy gate
4	The Evidence Plane	20%	10	Discovery and the gap map, activation, Firestore store, locked reports bucket, collector
5	Reporting & Narrative	15%	7	POA&M, SAR, OSCAL SSP from Firestore only; reporter identity; digest guardrails
6	Enforcement & Operations	10%	5	Remediation ladder, remediator identity, OPA gate, drift
7	Capstone	10%	5	Requirements, rubric, failure modes, submission

The Capstone Is Required

CGE-GCP certifies builders, so the credential takes both parts: the exam and a graded capstone. The capstone is the pipeline itself: the foundation, stages 1 to 3, at least two reports generated from Firestore only, and Stage 6 remediation in report_only mode or higher, deployed as code in your own project and submitted as a public repo link. It is graded pass/fail by an automated grader against five published dimensions, each weighted 20%, with a pass at a weighted average of 70 or higher and no auto-fail triggers. You get 2 attempts with a 24-hour cooldown, and appeals go to hello@grcengclub.com. Stage 5, the AI narrative, is optional extra credit and never graded. Read the CGE-GCP Capstone Overview and the published rubric at cert.grcengclub.com/rubric/cge-gcp before you build.

Scoring and Certification

Exam Score: 70% or higher (35 of 50) passes. Scenario questions score as individual questions, weighted equally with multiple choice.

Certification Awarded: Passing the exam and the capstone earns the Certified GRC Engineer, GCP Specialty (CGE-GCP) credential, valid for 365 days. Renew annually with an active Club membership (auto-renews) or 30 CPE hours.

DOMAIN 1: GCP FOUNDATIONS

15% of Exam, Approximately 8 Questions

What You Need to Know

Domain 1 gives you the Google Cloud mental model: the hierarchy, the identity system, and the responsibility boundary. Once you can read the resource hierarchy, scoping an audit gets much easier, because the hierarchy is a map of where each control is set and what it covers.

1.1 The GCP Mental Model: Organization, Folders and Projects (01_01)

Four levels, top to bottom: the Organization (tied to a domain), folders (which nest), projects, and the resources inside projects. Policy flows down: grant a role at a folder and every project under it inherits the grant. A control set at the top covers everything below it, and a control set on one project covers one project, so the first scoping question is where each control is set. The project is the unit that matters most: billing attaches to it, APIs are enabled per project, and IAM, quotas, and audit logs all live there. Billing accounts are a separate object with their own permission set, and one can pay for many projects. A free-trial account gives you projects with no Organization above them, so this course is project-scoped: every control in the pipeline is set at the project level. Inside a real Organization the pipeline works unchanged, and the appendix shows what you gain.

1.2 IAM, Service Accounts and Workload Identity (01_02)

Every IAM question has three parts. Who: the principal, a user, a group, or a service account. What: a role, a bundle of permissions. Where: the resource the binding is attached to. Together they form an allow policy binding, and every access review on Google Cloud is reading those. Roles come in three kinds. Basic roles (Owner, Editor, Viewer) grant thousands of permissions across nearly every service. Predefined roles are scoped to a service and a job. Custom roles hold the permissions you pick. A pipeline identity holding Editor is a least-privilege failure however well the rest is built. Service accounts give workloads identity: attach one to a function and the platform hands the workload short-lived tokens, so nobody holds a secret. A service account key is a JSON file that works from anywhere and never expires on its own; treat any user-managed key as a finding. Three more tools: IAM conditions narrow a grant, deny policies win over any allow, and Workload Identity Federation lets GitHub Actions exchange its own OIDC token for short-lived Google credentials, so CI stores no key.

1.3 Shared Responsibility and What You'll Engineer (01_03)

Google secures the data centers, the hardware, the hypervisor, and the managed services themselves. You secure everything you configure: who has access, which APIs are on, how buckets are exposed, where logs go, and whether anyone can prove any of it. Google calls its version shared fate, meaning it ships secure defaults and blueprints to help with your side. The configuration is still yours, and so is the evidence. Each pipeline stage covers part of your side: discovery tells you what is configured, activation closes the gaps in what is monitored, the evidence store proves what was true and when, reporting turns that proof into what an auditor reads, and enforcement keeps configuration from drifting back.

1.4 Lab 1: Stand Up Your GCP GRC Sandbox (01_04)

The sandbox every later lab runs in: one project, a $25 budget with alert thresholds (your first piece of evidence, a cost guardrail configured before the first resource exists), both logins, a pinned project, and your fork of the starter repo. Then run labs/00-setup/probe.sh. It confirms there is no Organization above the project, enables the APIs Lab 2 uses, and checks that billing is linked, Application Default Credentials are present, and Asset Inventory answers queries. On a new project an API can take a few minutes to finish enabling; wait two minutes and run the probe again. Your evidence for this lab is the probe output and the budget configuration, both reproducible.

Key Terms and Definitions

Organization: The top of the hierarchy, tied to a domain; a personal @gmail.com account has none.

Project: The unit of billing, enabled APIs, IAM, quotas, and audit logs; the scope of every control in this course.

Allow Policy Binding: Principal plus role plus resource: who can do what, where.

Basic Role: Owner, Editor, or Viewer; thousands of permissions across nearly every service. A finding on a service account.

Service Account: The identity a workload runs as; attached to the workload, which receives short-lived tokens.

Service Account Key: A downloadable JSON credential that works from anywhere and never expires on its own. A finding.

Workload Identity Federation: CI exchanges its own OIDC token for short-lived Google credentials; no key is stored.

Application Default Credentials: The local credentials Terraform and the Python client libraries use, created by the second login.

Practice Prompts (no answers here; the graded bank is separate)

1. An auditor asks for the system boundary of a workload that lives in one project inside a folder. A bucket guardrail is set on the folder and a role grant is set on the project. State what each control covers.

2. A binding grants roles/editor to a service account at the project level. Name the three parts of the binding, say why it is a finding, and say what kind of role should replace it.

3. A command against your lab project fails with USER_PROJECT_DENIED even though you own the project. What is misconfigured, and which command do you run to check before every lab?

DOMAIN 2: THE GCP GRC TOOLKIT

15% of Exam, Approximately 7 Questions

What You Need to Know

Eight services run this pipeline, and each has one job. Learn the job and you know when to reach for the service and which one owns an auditor's question. This domain covers the three you will read artifacts from on the exam, Cloud Asset Inventory, IAM, and Cloud Audit Logs, plus the way the course builds guardrails when the platform's own are out of reach.

2.1 The Services That Run the Pipeline (02_01)

Cloud Asset Inventory answers what exists and how it is configured: every resource and every IAM policy in the project through one search API. Cloud Audit Logs answer who did what. IAM decides who can act. The policy gate decides what can be deployed: every pull request's Terraform plan is checked against rules you write. Cloud Run functions run the collector, the report generators, and the remediator, triggered by Cloud Scheduler. Firestore stores your check results as documents you own. Cloud Storage holds the generated reports in a bucket with a locked retention policy. Vertex AI, with a Gemini model, writes the optional weekly narrative. Two Google services do posture and guardrails at the platform level, Security Command Center and Org Policy. Both need an Organization above your project, so the course builds the same controls itself and covers those two in the appendix.

2.2 Cloud Asset Inventory and Checks as Code (02_02)

gcloud asset search-all-resources lists every resource in the project with its type and location. search-all-iam-policies returns every IAM binding; ask it which bindings include allUsers and you have found every public resource with one query. The inventory tells you what exists. A check tells you whether it is acceptable. The pipeline ships five checks as code: CHK-STG-01 no public buckets, CHK-STG-02 uniform bucket-level access on, CHK-IAM-01 no basic roles on service accounts, CHK-IAM-02 no user-managed service account keys, and CHK-LOG-01 Data Access logs on for Cloud Storage. Each is a few lines of Python reading inventory data, mapped to NIST CSF 2.0 and SP 800-53. Read a result like an auditor: the check ID names the rule, the resource names exactly what, the status is pass or fail, the evidence is the inventory record the decision was based on, and the controls list the mappings. Because the evidence travels with the result, anyone can re-check the decision later. One timing fact: the inventory updates shortly after a change. On the validation project a newly public bucket appeared in search-all-iam-policies 57 seconds later.

2.3 Preventive and Corrective Controls on Google Cloud (02_03)

IAM controls who can act. A guardrail controls what can exist, whoever is asking. Google's platform guardrail is Org Policy: constraints enforced at request time, with a native dry-run mode. It needs an Organization. On a project with none, even the Owner gets permission denied, and the Org Policy Admin role cannot be granted on a project at all. So the course builds the control in three layers you own. Layer one, preventive: secure defaults in Terraform, so every bucket the pipeline creates has public access prevention enforced and uniform access on. Layer two, preventive: the policy gate, which blocks a public bucket, a basic role, or a service account key before it reaches the project. Layer three, corrective: a remediation function that fixes what slipped past. It starts in report-only mode, then approval mode, then automatic, and you move up only when the data from the mode below says it is safe. Observe, approve, automate.

2.4 Cloud Audit Logs: The Raw Material of Evidence (02_04)

Four kinds of audit logs. Admin Activity logs record configuration changes and are always on. System Event and Policy Denied logs are also written by default. Data Access logs, which record reads and data writes, are mostly off by default; turning them on is a decision you make per service, and the pipeline makes it in code so

the decision is recorded. Logs land in log buckets. The _Required bucket holds Admin Activity and System Event logs for 400 days, and you cannot change that. The _Default bucket holds the rest for 30 days unless you change it. If an auditor asks for a year of evidence and your Data Access logs live in a 30-day bucket, you have a gap, and a log sink to longer storage is the fix. In Logs Explorer, filter on protoPayload.methodName to find changes with who made them and when. A saved query is a repeatable test, and a repeatable test is evidence.

2.5 Lab 2: Turn It All On (02_05)

The only lab where you click. Turn on Data Access logs for Cloud Storage in the console, create a test bucket, and grant allUsers the Storage Object Viewer role. The grant succeeds: a new project with no Organization does not block public buckets, which is why the checks exist. Find the bucket with the Asset Inventory allUsers query (wait a minute if it comes back empty), run CHK-STG-01 from the repo and watch it fail, then find who did it in Logs Explorer with the storage.setIamPermissions method. Remove the grant and run the query and the check again. Detected, fixed, confirmed: the loop the whole pipeline automates. Keep the Data Access setting, and delete the test bucket.

Key Terms and Definitions

Cloud Asset Inventory: One search API over every resource and IAM policy in the project; the collector's data source.

Check: A rule as code with an ID, evaluated against inventory data; the result carries its own evidence.

Preventive Control: Stops a bad change before it exists: secure defaults and the policy gate.

Corrective Control: Fixes what slipped past: the remediation function.

Admin Activity Log: The always-on record of configuration changes; kept 400 days in the _Required bucket.

Data Access Log: The record of reads and data writes; mostly off by default and enabled per service.

Log Sink: A route from audit logs to a log bucket with longer retention.

Org Policy: Google's request-time guardrail; needs an Organization, so it is appendix material here.

Practice Prompts (no answers here; the graded bank is separate)

1. An assessor asks whether any resource in the project is public. Which single query answers that, and why might it come back empty a few seconds after a change?

2. Your Data Access logs for Cloud Storage live in the _Default bucket and the audit period is twelve months. State the gap and the fix.

3. A teammate proposes an Org Policy constraint to block public buckets in a free-trial project. Explain why that fails and name the three layers the course uses.

DOMAIN 3: TERRAFORM ON GCP

15% of Exam, Approximately 8 Questions

What You Need to Know

You already write Terraform; this domain covers the google provider and puts governance itself in code. By the end of Lab 3 the foundation is deployed, six identities exist with nothing granted, your fork's CI is connected without keys, and you have watched the gate fail on purpose.

3.1 The google Provider: Auth, APIs and State (03_01)

Two providers: google, and google-beta for features still in preview. Locally the provider reads the Application Default Credentials you created in Lab 1. In CI it uses Workload Identity Federation. In neither case is there a key file. APIs are infrastructure too: google_project_service enables each service's API in code, which also documents exactly which services the system uses. Set disable_on_destroy to false, or destroying one stage can switch off an API another stage still depends on. State goes in a Cloud Storage bucket; the gcs backend locks state natively. The bootstrap creates the state bucket with versioning on and public access prevention enforced, because the state file describes your whole environment. Each stage is its own root module with its own state, so a mistake in Stage 6 cannot touch the evidence store.

3.2 Governance as Code: IAM and Guardrails in Terraform (03_02)

Terraform has three ways to manage project IAM, and one can lock you out of your own project. google_project_iam_member adds a single member to a role and touches nothing else. google_project_iam_binding owns every member of one role. google_project_iam_policy owns the entire policy and removes anything not in your code, including your own access. The course uses iam_member, always. Guardrails start as secure defaults: every bucket sets public_access_prevention to enforced and uniform_bucket_level_access to true. The gate makes the defaults mandatory: a Rego rule reads the plan and fails the pull request if a bucket is missing either setting, or if any binding grants a basic role or creates a service account key. Before every apply, read the plan like an auditor. The plan is the change ticket.

3.3 Lab 3: Deploy Your Foundation (03_03)

Run the bootstrap to create the state bucket and the backend config. Apply stage 00-foundation: it enables the APIs and creates six service accounts, one per pipeline job (grc-builder, grc-collector, grc-reporter, grc-narrator, grc-remediator, grc-scheduler), each with zero permissions. Prove it from the project IAM policy. Then run arm-your-fork.sh with your fork as the argument. It is required and has no default, because a default would make your project trust someone else's repository. The script creates a Workload Identity Federation pool that trusts tokens from your fork only, and you set two repository variables (variables, since there is nothing secret to store). CI plans; it never applies. Open a pull request and watch the gate pass. Then add a binding granting roles/editor to the collector and watch it fail and name the rule. A gate you have never seen fail is a gate you cannot trust.

Key Terms and Definitions

google_project_service: Enables an API as code; set disable_on_destroy = false.

gcs Backend: Terraform state in a Cloud Storage bucket with native locking.

Additive IAM: google_project_iam_member: adds one member to one role and touches nothing else.

Authoritative IAM: _iam_binding and _iam_policy: own a whole role or the whole policy and remove what is not in code.

Secure Defaults: public_access_prevention = "enforced" and uniform_bucket_level_access = true on every bucket.

Policy Gate: conftest checking each pull request's plan against the Rego rules in policy/.

Practice Prompts (no answers here; the graded bank is separate)

1. A tutorial tells you to set GOOGLE_APPLICATION_CREDENTIALS to a downloaded JSON key so Terraform can run in CI. Name the keyless alternative for CI and the one for your laptop.

2. A plan replaces several google_project_iam_member resources with one google_project_iam_policy. What could the apply remove, and what should the reviewer ask?

3. Why does arm-your-fork.sh refuse to run without a fork argument? Describe what a default value would have trusted.

DOMAIN 4: THE EVIDENCE PLANE

20% of Exam, Approximately 10 Questions

What You Need to Know

The heaviest domain on the exam, by design: stages 1 through 3 of the pipeline are where evidence is born. Discovery measures before anything acts, activation enables only what is missing, and the evidence store holds every check result once, with lineage, next to reports that cannot be deleted.

4.1 Stage 1, Discovery: Detect What's Running (04_01)

Stage 1 has one rule: change nothing. Discovery asks four questions. Are Data Access logs on for Cloud Storage? Does the audit log sink exist, and did the pipeline make it? Which buckets are missing a guardrail setting? Which service accounts have user-managed keys? One short script answers them with read-only gcloud calls, and Terraform runs it as an external data source, so the answers land in state as the gap map. Each item reads present (leave it), absent (activate it), different (a human decides, because someone may have chosen that setting on purpose), or pipeline (the pipeline created it, keep managing). Run it twice and you get the same answer, which makes it safe to run against any project on day one.

4.2 Stage 2, Activation: Enable What's Missing (04_02)

Activation reads the gap map and acts on the absent items only. The pattern in code is a count driven by the gap map. If Data Access logs are absent, activation creates a google_project_iam_audit_config. If they are present, Terraform adopts that setting and changes nothing. If they are different, the count is zero and activation leaves them alone. It also creates a log bucket, grc-audit-logs, with 365-day retention, and a sink, grc-audit-sink, that routes audit logs into it, because the _Default bucket keeps 30 days and an audit period is usually longer. Apply, then run discovery again: the sink now reads pipeline. Run activation a second time and the plan says no changes. If something drifts, the plan diff is your drift report.

4.3 Stage 3, The Evidence Store: Firestore, Bucket Lock and the Collector (04_03)

The inventory shows you today; an auditor asks about last March. Stage 3 runs your checks on a schedule and writes every result into a store you own. Firestore holds three collections: assessments (one document per check per resource, with the status, the evidence record, and when it was collected), frameworks (the controls of NIST CSF 2.0), and mappings (the crosswalk from each check to CSF and on to SP 800-53). Every assessment document ID is built from the check and the resource, so the same result collected twice updates one document. The collector is a Cloud Run function that Cloud Scheduler calls daily. It runs as grc-collector, which can read Asset Inventory and write Firestore, nothing else. The function builds as grc-builder, and grc-scheduler holds the invoker role on that one function. Reports go to a Cloud Storage bucket with a retention policy, and then the policy is locked. Bucket Lock is permanent: nobody, including you as Owner, can delete or replace an object until it ages out, and the lock can never be removed. The labs use a one-day window; in production you match your audit retention requirement.

4.4 Lab 4: Evidence Flowing End to End (04_04)

Apply discovery, activation, and the evidence store in order, seed the frameworks and mappings, and force-run the collector from Cloud Scheduler. One check is already failing and you did not cause it: CHK-IAM-01. On a project with no Organization, Google gives two default service accounts the Editor role. The pipeline uses neither, so you remove both grants, collect again, and watch it pass. Prove the upsert: note the document count, collect again, count again. Same number, and every document carries the newest run_id. Then try to break the evidence: upload a test file to the reports bucket and try to delete it. The delete fails and the error names the retention policy. Save that error in your evidence folder. That failed delete is your WORM proof: write once, read many.

Key Terms and Definitions

Gap Map: Discovery's Terraform output: each baseline item as present, absent, different, or pipeline.

Conditional Activation: A count driven by the gap map, so Terraform acts only on what is absent.

Ownership Marker: The managed-by: cgegcp label on what activation created, so later runs keep managing it.

Deterministic Document ID: Check plus resource; a re-run updates one document, so counts stay honest.

Lineage: run_id and collected_at stamped on every document by every collector run.

Bucket Lock: A locked retention policy: permanent, and binding on the Owner too.

WORM Proof: The captured failed delete whose error names the retention policy.

Collect Once: One set of check results feeding every framework through the mappings data.

Practice Prompts (no answers here; the graded bank is separate)

1. A gap map reads audit_sink = "pipeline" and data_access_storage = "different". For each, say what activation does on the next apply and why.

2. A collector writes each result under a random document ID. What goes wrong with report counts over a month, and what ID strategy plus which two fields fix it?

3. An assessor asks you to prove last month's POA&M was not replaced after it was written. Which property of the reports bucket makes the case, and what failed operation do you show?

DOMAIN 5: REPORTING & NARRATIVE

15% of Exam, Approximately 7 Questions

What You Need to Know

Stages 4 and 5 turn stored evidence into the documents an assessor reads, plus an optional weekly digest for people who will never open a POA&M. The load-bearing rules: reports read from Firestore only, the reporter is a different identity from the collector, and AI describes, never decides.

5.1 Stage 4, Reporting: ATO Deliverables from Firestore Only (05_01)

Reports read from the evidence store and nothing else. No report function calls Asset Inventory directly. If the POA&M and the SAR both come from Firestore, they agree with each other every time, and you can show which collected evidence produced each line. Three reports: a POA&M, daily, listing every open finding with its control mapping, severity, and days open; a SAR, weekly, summarizing assessment results by control with the CSF 2.0 to SP 800-53 crosswalk; and an OSCAL 1.1.2 SSP, weekly and machine-readable. Each lands in the locked bucket with a timestamp in the name. The reporter runs as grc-reporter: it can read Firestore and create objects in the reports bucket. It cannot write Firestore, and the collector cannot write reports. One identity records the facts, a different identity writes the reports from them. That is separation of duties, enforced by IAM. Because the bucket is locked, every report version is retained: nobody can regenerate history to look better.

5.2 Lab 5: Generate Your First ATO Deliverables (05_02)

Apply stage 04 and force-run the three report jobs. Open the POA&M, take the ID of any line, and open that document in Firestore: same resource, same run_id, with the inventory record in the evidence field. That trace is what you walk an auditor through. Validate the SSP with the script in the lab folder against the official NIST OSCAL 1.1.2 SSP schema. Then prove separation of duties. Even as project Owner, you cannot act as the reporter until you grant yourself that right on this one identity. Grant it, try to write a Firestore document as the reporter, and capture the PERMISSION_DENIED response. Remove the grant. Save the error next to your WORM proof: two failed actions, two pieces of evidence that your controls work.

5.3 Stage 5, AI Narrative: Describe, Never Decide (05_03)

Stage 5 is optional by design. A Gemini model on Vertex AI writes a weekly digest: a plain-language summary of what changed in your posture. The model reads the week's SAR and POA&M, which are already generated from evidence. It never reads raw findings, never assigns severity, never closes an item, and never changes anything. Its service account, grc-narrator, can read the reports bucket, add the digest to it, and call Vertex AI, nothing more. The prompt tells the model to summarize only what is in the provided reports and to cite the file and line for every factual sentence, and the output is labeled AI-generated. Delete Stage 5 and the pipeline still works: evidence is collected, reports are generated, enforcement runs. Stage 5 is extra credit in the capstone and never graded.

Key Terms and Definitions

POA&M: Plan of Action and Milestones: every open finding with control mapping, severity, and days open. Daily.

SAR: Security Assessment Report: assessment results summarized by control. Weekly.

OSCAL SSP: The machine-readable System Security Plan, OSCAL version 1.1.2. Weekly.

Firestore-Only Rule: Report functions read the evidence store, never live platform APIs.

Separation of Duties: The collector writes facts, the reporter writes reports, and neither can do the other's job.

Describe, Never Decide: The AI guardrail: the model summarizes finished reports and changes nothing.

Practice Prompts (no answers here; the graded bank is separate)

1. A teammate makes the SAR fresher by calling Asset Inventory at render time. What did the change break, and how would a reviewer catch it in the code?

2. You are project Owner and impersonating grc-reporter fails with a getAccessToken permission error. Is that a misconfiguration or a control? Explain.

3. A proposal lets the digest model reorder POA&M items by predicted risk. Does that describe or decide? Say which line of the Stage 5 rule it crosses.

DOMAIN 6: ENFORCEMENT & OPERATIONS

10% of Exam, Approximately 5 Questions

What You Need to Know

Stage 6 closes the loop: detected, fixed, re-collected, documented. Enforcement is a ladder you climb one rung at a time, through an identity that can do exactly one job. Operations keeps the pipeline honest: a gate on every pull request and a nightly drift check.

6.1 Stage 6, Enforcement: Close the Loop (06_01)

An automated fix that breaks production loses the room for a year, so enforcement climbs a ladder. Rung one: report_only. Rung two: approve, a fix on a human's yes. Rung three: auto. You move up only when the data from the rung below says it is safe. The reference control is CHK-STG-01: no Cloud Storage bucket may be public. The remediator runs after each collection and reads the failing results from the evidence store. In report_only mode it writes down what it would have done and changes nothing. In approve mode it writes an approval request; a human approves, and the function removes the public grant and enforces public access prevention on that one bucket. The mode is one Terraform variable, changed through a pull request, so the decision to automate is itself on the record. grc-remediator holds a custom role, grcBucketRemediator, with four bucket permissions. It cannot read or delete objects, and it cannot delete a bucket. The policy gate is the preventive half, with four rules: no basic roles, no service account keys, bucket guardrails on every bucket, and the remediation function must run as its own dedicated service account. drift.yml runs the plan nightly and opens an issue if reality and code disagree.

6.2 Lab 6: Close the Loop (06_02)

Apply stage 06 in report_only mode, then make a test bucket public by hand, outside Terraform, the way it happens in real life. Collect, then run the remediator: CHK-STG-01 fails, and the remediations collection records would_fix with the bucket still public. Raise the mode to approve through a pull request, run the remediator again, and approve the pending fix with the script in the lab folder. The next run records fixed, the next collection passes, and the next POA&M drops the line. Make the bucket public again and the case reopens: an old fixed record never hides a new failure. Then prove the preventive half: a pull request adding a bucket without guardrails and a service account key fails the gate, which names each rule. Save the failures in your evidence folder. Teardown runs in reverse stage order; the locked reports bucket stays until its one-day retention expires.

Key Terms and Definitions

Remediation Ladder: report_only, approve, auto: each rung earned by the data from the one below.

report_only: The remediator records what it would fix (would_fix) and changes nothing.

approve: The remediator requests approval and fixes only after a human says yes.

grcBucketRemediator: The remediator's custom role: get, getIamPolicy, setIamPolicy, and update on buckets.

Policy Gate: conftest and OPA on every pull request's plan; the preventive half of enforcement.

Drift Detection: drift.yml plans nightly and opens an issue when reality and code disagree.

Practice Prompts (no answers here; the graded bank is separate)

1. A team wants to deploy the remediator straight into auto mode to save time. What evidence should exist before each rung, and how is the change to auto recorded?

2. A reviewer finds the remediation function running as a default service account. Which gate rule should have blocked it, and what does the reviewer conclude about blast radius?

3. Describe the evidence trail one approved fix leaves behind, from the failing check to the POA&M that no longer lists it.

DOMAIN 7: CAPSTONE

10% of Exam, Approximately 5 Questions

What You Need to Know

The capstone is the credential's second half and it is required: your pipeline, in your project, from your repo, graded automatically against a rubric published from day one. Domain 7's exam questions test whether you know what the rubric demands and how the grader checks it.

7.1 Capstone Brief: Ship Your Pipeline (07_01)

The requirements: the foundation; stages 1 to 3 with the evidence store and its locked reports bucket; at least two reports generated from Firestore only, with real run history; Stage 6 remediation in report_only mode or higher; the gate and drift workflows, armed on your fork; docs/CONTROLS.md current with your code; a README that opens with your tenancy statement; and an evidence folder with your WORM proof, your separation-of-duties proof, and a gate failure from a closed test pull request. Five dimensions, each 20%: Infrastructure-as-Code Quality, Control Implementation and Identity Design, Evidence Integrity and Traceability, Pipeline Operations, and Documentation and Control Mapping. Pass is a weighted average of 70 or higher with no auto-fail triggers. Auto-fails: a private or inaccessible repo, any stored credential, real PII or production credentials, an unmodified or trivially modified fork of the starter, or no README. Grading is automated with 2 attempts and a 24-hour cooldown, and appeals go to hello@grcengclub.com.

7.2 Capstone Walkthrough: The Reference Build (07_02)

A complete passing capstone in a fresh project. The README's first line states the tenancy, and a table lists every service account, what it can do, and which stage uses it. The trace a reviewer will do: pick a POA&M line, find the Firestore document by its ID, find the asset record it came from, and find the collector run that wrote it in the logs. The failures seen most: a service account with a basic role left over from debugging, a reports bucket without the lock, reports that call Asset Inventory directly, and a README with no tenancy statement. The gate or ./self-check.sh catches all four if you run them.

7.3 Course Wrap-Up and What's Next (07_03)

Taking it to work: run discovery first, read-only, against a real project, and show the gap map to your team before you activate anything. Run remediation in report_only mode and share what it would have fixed before you let it act. If your company has an Organization, read the appendix: Org Policy and Security Command Center apply these controls across every project at once.

7.4 Certification Exam Guide (07_04, article)

Logistics: scheduling through the Academy portal, the domain weights with question counts, scenario strategy (read the artifact first and the question second), and sequencing. Take the exam while the concepts are fresh, then submit the capstone, so your scheduled collector keeps building the run history the grader looks for. Stay inside your free-trial window, because the evidence has to be real.

Key Terms and Definitions

Rubric Dimension: One of five published grading axes, each weighted 20%.

Auto-Fail Trigger: A condition that fails a submission regardless of score, such as a stored credential or no README.

Tenancy Statement: The README's first line: a project with no Organization, or a project inside one.

Run History: Scheduled executions accumulated over time; the grader looks for it and a burst of manual runs does not count.

Evidence Folder: evidence/ in your repo: the WORM proof, the separation-of-duties proof, and a gate failure.

self-check.sh: The mechanical half of the rubric, run locally before you submit.

Practice Prompts (no answers here; the graded bank is separate)

1. A submission has strong code, but every scheduled job ran once, by hand, the night before. Which dimension takes the hit, and what would have prevented it?

2. List three auto-fail triggers and, for each, the check you run before submitting that catches it.

3. Your exam is next week and your collector has been running daily for three days. Order the exam and the capstone submission and defend the timing.

EXAM DAY AND NEXT STEPS

Read each scenario artifact before the question: say what the IAM binding, Terraform plan, audit log entry, inventory record, check result, or run log does in your own words, then read what is being asked. Manage time across 50 questions in 75 minutes. Submit the capstone while your pipeline's run history is live, and run ./self-check.sh first.

After you pass both parts, the credential is valid for 365 days. Renew annually through an active GRC Engineering Club membership (auto-renews) or 30 CPE hours. Then take the pattern to work: start with discovery, and build from there.

Website: www.grcengclub.com **Email:** academy@grcengclub.com **Community:** GRC Engineering Club (Patreon)