GCP Professional Cloud DevOps Engineer

Complete Study Guide

Professional Certification Guide

✓ Complete exam coverage
✓ Structured learning path
✓ Real-world scenarios
✓ Quick reference tables
✓ Best practices & tips

Version 1.0
Generated: December 22, 2025
© 2025 Study Materials

Table of Contents

GCP Professional Cloud DevOps Engineer — Study Guide

Purpose: Topic-by-topic plan aligned to the official domains, emphasizing SRE practices, CI/CD, observability, performance, and cost optimization.

Exam Snapshot

- Format: 50–60 questions, 2 hours
- Focus: Org bootstrap, SRE, CI/CD across app/infra/ML, observability/troubleshooting, performance and cost
- Core services: Cloud Build, Cloud Deploy, Artifact Registry, GKE, Cloud Run, Cloud Functions, Terraform (infra as code), Cloud Monitoring/Logging/Trace, Error Reporting, Profiler, Debugger, Pub/Sub, VPC, IAM, Config Sync/Policy Controller

Domain 1 — Bootstrap and maintain a Google Cloud organization (Medium–High)

Topics and objectives:

- Org structure: Org → Folders → Projects; least-privilege IAM; service accounts; workload identity
- Billing setup, budgets/alerts, quota and limit management
- Networking foundations: VPC design (shared VPC, private service connect), subnetting, firewall rules, Cloud NAT
- Organization Policy constraints, CMEK requirements, regional policy guardrails
- Policy and config management: Config Sync/Policy Controller (OPA/Gatekeeper)

Hands-on:

- Create org-level budgets and alerts; set quotas
- Configure Shared VPC with isolated projects for build, staging, prod
- Apply Policy Controller constraint to block public services

Quick check:

- When to use Shared VPC vs per-project VPC?

Domain 2 — Apply SRE practices (High)

Topics and objectives:

- SLI/SLO/SLA fundamentals; error budgets; burn rate alerts
- Reliability patterns: graceful degradation, overload protection, circuit breakers, autoscaling policies
- Incident response: severity, paging, runbooks, postmortems/blameless culture
- Capacity planning and load testing; release policies to protect error budgets

■ SLI/SLO/SLA Deep Dive

Definitions:

- **SLI (Service Level Indicator):** Quantitative measure of service level
- Examples: Availability %, latency p95, error rate, throughput
- **SLO (Service Level Objective):** Target value for an SLI
- Example: 99.9% availability over 30 days
- **SLA (Service Level Agreement):** Business contract with consequences
- Example: 99.9% availability or customer gets refund

Critical Relationship:

```
SLA (Customer promise) ≥ SLO (Internal target) > Actual performance

Example:
SLA: 99.5% (promise to customers)
SLO: 99.9% (internal target, buffer for safety)
Actual: 99.95% (current performance)
```

■ Error Budget Calculation

Formula:

```
Error Budget = 1 - SLO

Example: 99.9% availability SLO
Error Budget = 1 - 0.999 = 0.001 = 0.1%
```

Time-based Error Budget (30 days):

```
30 days = 30 × 24 × 60 = 43,200 minutes

99.9% SLO:
Allowed downtime = 43,200 × 0.001 = 43.2 minutes/month

99.5% SLO:
Allowed downtime = 43,200 × 0.005 = 216 minutes/month = 3.6 hours

99.99% SLO:
Allowed downtime = 43,200 × 0.0001 = 4.32 minutes/month
```

Request-based Error Budget:

```
1 million requests/day × 30 days = 30M requests/month

99.9% SLO:
Allowed failures = 30M × 0.001 = 30,000 failed requests/month
= 1,000 failures/day
= ~42 failures/hour
```

■ Multi-Window, Multi-Burn-Rate Alerts

Concept: Alert at different severities based on how fast you're burning error budget

Burn Rate = How many times faster than normal you're using error budget

Example for 99.9% SLO (0.1% error budget):

Window	Burn Rate	Error Rate	Action
5 minutes	14x (1.4%)	1.4%	■ Page
1 hour	4x (0.4%)	0.4%	■ Page
6 hours	2x (0.2%)	0.2%	■ Ticket
3 days	1x (0.1%)	0.1%	■ Monitor

Why multiple windows?

- **Short window (5 min):** Catch severe incidents fast
- **Long window (1 hr):** Avoid false positives from transient spikes
- **Both must fire:** Confirms it's a real issue, not a blip

Cloud Monitoring Alert Policy Example:

```
conditions:
- displayName: "Fast burn (5min @ 14x)"
  conditionThreshold:
    filter: |
      resource.type="cloud_run_revision"
      metric.type="run.googleapis.com/request_count"
    aggregations:
    - alignmentPeriod: 300s
      perSeriesAligner: ALIGN_RATE
    comparison: COMPARISON_GT
    thresholdValue: 0.014  # 1.4% error rate
    duration: 300s

- displayName: "Sustained burn (1hr @ 4x)"
  conditionThreshold:
    filter: |
      resource.type="cloud_run_revision"
      metric.type="run.googleapis.com/request_count"
    aggregations:
    - alignmentPeriod: 3600s
      perSeriesAligner: ALIGN_RATE
    comparison: COMPARISON_GT
    thresholdValue: 0.004  # 0.4% error rate
    duration: 3600s

combiner: AND  # Both conditions must be true
```

■ Real Incident Response Scenario

> **Scenario:** 3 AM page - "API latency SLO breach"

Step 1: Acknowledge & Assess (2 min)

```
# Check current status
gcloud monitoring dashboards list

# Quick metrics
gcloud monitoring time-series list \
  --filter='metric.type="run.googleapis.com/request_latencies"' \
  --interval-start-time='2024-01-01T03:00:00Z'
```

Step 2: Triage Severity (3 min)

```
Severity Matrix:
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■ Impact             ■ Severity           ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■ Service down       ■ P0 (Critical)      ■
■ > 50% users impact ■ P1 (High)          ■
■ Degraded perf      ■ P2 (Medium)        ■
■ Minor issue        ■ P3 (Low)           ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■

This incident: p95 latency 5s (SLO: 500ms) = P1
```

Step 3: Investigate (10 min)

```
# Check recent deploys
gcloud run revisions list --service=api --region=us-central1

# Check logs for errors
gcloud logging read \
  'resource.type="cloud_run_revision" severity>=ERROR' \
  --limit=50 --format=json

# Check Cloud SQL connections
gcloud sql operations list --instance=main-db --limit=10

# Check traces for slow operations
gcloud trace list --limit=10
```

Step 4: Mitigate (5 min)

```
# Option 1: Rollback to previous revision
gcloud run services update-traffic api \
  --to-revisions=api-v123=100 --region=us-central1

# Option 2: Scale up if capacity issue
gcloud run services update api \
  --max-instances=100 --region=us-central1

# Option 3: Restart if stuck
gcloud run services update api \
  --update-env-vars=RESTART="$(date)" --region=us-central1
```

Step 5: Verify (5 min)

```
# Check metrics improved
# Monitor for 5-10 minutes
# Confirm SLO back in range
```

Step 6: Document (30 min post-incident)

- Timeline of events
- Root cause (DB query N+1 problem in new code)
- Impact (2.3% of monthly error budget consumed)

- Action items (add query optimization test)

■ Postmortem Template

```
# Incident Postmortem: API Latency Spike

Date: 2024-01-15
Duration: 23 minutes
Severity: P1

## Impact
- 23 minutes of degraded performance
- p95 latency: 5000ms (SLO: 500ms)
- 0.05% of monthly error budget consumed
- ~1,000 customer requests affected

## Timeline (UTC)
- 03:14: Alert fired
- 03:16: On-call engineer acknowledged
- 03:20: Identified new deployment as cause
- 03:25: Rolled back to previous version
- 03:30: Latency returned to normal
- 03:37: Incident closed

## Root Cause
- New code introduced N+1 query problem
- Each API request made 100+ DB queries
- Load testing didn't catch (used test data with 1 record)

## What Went Well
■ Alert fired within 2 minutes
■ Rollback was smooth and automated
■ Communication was clear

## What Went Wrong
■ Load testing insufficient (test data not realistic)
■ No query count monitoring
■ Deploy happened at 3 AM (low-traffic time masked issue)

## Action Items
- [ ] Add query count monitoring (Owner: Dev, Due: 2024-01-20)
- [ ] Improve load test data (Owner: QA, Due: 2024-01-22)
- [ ] Add query count limit in staging (Owner: DevOps, Due: 2024-01-18)
- [ ] Change deploy window to daytime (Owner: Release, Due: Now)

## Lessons Learned
- Realistic test data is critical
- Monitor database query patterns
- Deploy during high-traffic for faster detection
```

Hands-on:

- Define SLIs (availability=99.9%, latency p95<500ms) for a service
- Calculate error budget: 99.9% = 43.2 min/month downtime allowed
- Implement multi-window burn-rate alert (5min@14x + 1hr@4x)

- Create runbook for common incident (DB connection exhaustion)
- Simulate incident: trigger alert, investigate, mitigate, document
- Configure autoscaling: HPA for GKE (CPU>70%), Cloud Run min-instances=5

Quick check:

- 99.95% SLO over 30 days = how much downtime? (21.6 minutes)
- What's a 10x burn rate for 99.9% SLO? (1% error rate vs 0.1% normal)
- When to page vs ticket? (Fast burn + sustained burn = page; slow burn = ticket)

Domain 3 — Build and implement CI/CD pipelines (High)

Topics and objectives:

- CI: Cloud Build steps, private pools, caching, parallelism, test/reporting, SAST/DAST integration
- CD: Cloud Deploy delivery pipelines, targets, approvals, rollbacks; progressive delivery (canary, blue/green)
- Infra pipelines: Terraform in Cloud Build; policy checks; drift detection
- ML pipeline basics: CI for model code, artifact/version registry, staged serving rollouts
- Supply chain security: provenance/SLSA, attestation, Binary Authorization (GKE)

Hands-on:

- CI pipeline: test → build → scan → push → provenance attest
- CD pipeline: dev → staging → prod with manual approval and canary step
- Infra pipeline: Terraform plan/apply with policy-as-code gate

Quick check:

- Compare blue/green vs canary for a stateful GKE workload

Domain 4 — Implement observability and troubleshoot (High)

Topics and objectives:

- Telemetry: metrics (managed + custom), logs, traces; log-based metrics and alerting
- Uptime checks, SLO dashboards, runbooks; on-call rotations
- Debugging: Error Reporting, Profiler, Debugger; tracing via OpenTelemetry
- Troubleshooting playbooks: high latency, elevated error rate, memory/cpu saturation, consumer lag

Hands-on:

- Create log-based metric for 5xx ratio; alert at burn-rate thresholds
- Add tracing to a multi-service app and visualize critical path
- Use Profiler to locate CPU hot path; fix and measure improvement

Quick check:

- How do you debug intermittent latency spikes end-to-end?

Domain 5 — Optimize performance and cost (Medium–High)

Topics and objectives:

- Right-sizing: requests/limits (GKE), concurrency (Cloud Run), autoscaling signals
- Caching layers: Redis/Memorystore, CDN (Cloud CDN) for static/HTTP
- Storage/DB tuning: indexes, query plans, connection pools, read replicas, partitioning/sharding
- Cost controls: committed use discounts, autoscaling to zero, coldline/archival storage, budget alerts

Hands-on:

- Tune Cloud Run concurrency and min instances; measure p95 improvements
- Add Cloud CDN to public endpoints; confirm cache hit ratio
- Introduce Redis cache for hot keys; reduce DB load

Quick check:

- Which metrics indicate over-provisioning vs under-provisioning?

High-Frequency Topic Map (What appears most)

- SLOs, SLIs, error budgets, multi-window burn-rate alerts — High
- Cloud Build + Cloud Deploy CI/CD, progressive delivery — High
- Observability stack, log-based metrics, tracing, prof/prod debugging — High
- Org bootstrap: IAM, policies, Shared VPC, budgets/quotas — Medium–High
- Performance/cost tuning: autoscaling, caching, CDN, DB tuning — Medium–High
- Supply chain security, Binary Authorization, provenance — Medium

---

30-Day Study Plan (2 hrs/day)

- Week 1: Org/IAM/VPC foundation, budgets/quotas, Policy Controller; hands-on guardrails
- Week 2: SRE fundamentals, SLOs/SLIs, burn-rate alerts; incident response/runbooks
- Week 3: CI/CD pipelines (apps + infra), canary/blue-green, supply chain security
- Week 4: Observability, troubleshooting drills, performance/cost tuning; final mocks

---

Useful Configs & Snippets

- Multi-burn-rate alert (concept):

```
Short window: 5 min @ 14x budget burn → page
Long window: 1 hr @ 4x budget burn → page
Ticket window: 6 hr @ 2x → ticket only
```

- Cloud Deploy pipeline (excerpt):

```
apiVersion: deploy.cloud.google.com/v1
kind: DeliveryPipeline
serialPipeline:
  stages:
  - targetId: staging
    profiles: [canary]
  - targetId: prod
    strategy:
      canary:
        runtimeConfig:
          cloudRun:
            percentTraffic: [10,50,100]
```

- Terraform in Cloud Build (plan/apply gates):

```
steps:
- name: 'hashicorp/terraform:1.6.0'
  entrypoint: 'bash'
  args: ['-c','terraform init && terraform plan -out=tfplan']
- name: 'gcr.io/cloud-builders/gcloud'
  args: ['artifacts','attestations','evaluate','--policy=policy.yaml','--
artifact=tfplan']
- name: 'hashicorp/terraform:1.6.0'
  entrypoint: 'bash'
  args: ['-c','terraform apply -auto-approve tfplan']
```

Final Review Checklist

- [ ] Org structure, Shared VPC, policy/guardrails clear
- [ ] SLOs/SLIs + burn-rate alerts implemented
- [ ] Robust CI/CD with progressive delivery + rollback paths
- [ ] Observability: logs/metrics/traces dashboards + runbooks
- [ ] Troubleshooting playbooks for top failure modes
- [ ] Cost/perf tuning tactics with measurable outcomes