GCP Professional Cloud Developer

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

Table of Contents

GCP Professional Cloud Developer — Study Guide

Purpose: A structured, topic-by-topic plan aligned to the official exam domains, with focus indicators, hands-on tasks, and quick checks.

Exam Snapshot

- Format: 50–60 questions, 2 hours
- Focus: Design, build, deploy, integrate, secure, and observe applications on GCP
- Core services: Cloud Run, App Engine, GKE, Cloud Functions, Cloud Build, Artifact Registry, Pub/Sub, Cloud SQL/Firestore/Spanner, Memorystore, Cloud Logging/Monitoring, Secret Manager, IAM

Domain 1 — Design scalable, available, and reliable applications (High)

Topics and objectives:

- Workload placement: Cloud Run vs App Engine vs GKE vs Cloud Functions (latency, scaling, portability)
- Stateless vs stateful design; 12-factor app principles
- Resiliency patterns: retries, timeouts, circuit breakers, bulkheads
- Multi-region and regional HA patterns; load balancing options (Global HTTP(S) LB, Cloud Run domain mapping)
- Data design: choosing Cloud SQL vs Firestore vs Spanner; RTO/RPO; multi-region configs
- Event-driven architectures (Pub/Sub): at-least-once delivery, idempotency, ordering keys

■ Compute Service Decision Matrix

```
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■ Requirement        ■ Cloud Run    ■ App Engine   ■ GKE          ■ Functions    ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■ HTTP requests      ■ ■ Best      ■ ■ Good      ■ ■ Good      ■ ■■  Limited  ■
■ Background tasks   ■ ■ Jobs      ■ ■■  Tasks    ■ ■ CronJobs  ■ ■ Best      ■
■ Scale to zero      ■ ■ Yes       ■ ■ No (Std)  ■ ■ No        ■ ■ Yes      ■
■ Cold start         ■ ~1-3s        ■ ~10-30s      ■ N/A (warm)   ■ ~1-5s        ■
■ Max timeout        ■ 60 min       ■ 60 min       ■ Unlimited    ■ 9 min (2G)   ■
■ Stateful workloads ■ ■ No        ■ ■■  Limited  ■ ■ StatefulSet■ ■ No        ■
■ Custom runtime     ■ ■ Container ■ ■■  Limited  ■ ■ Any       ■ ■■  Runtimes ■
■ Pricing model      ■ Request-based■ Instance-hr  ■ Node-hr      ■ Invocation   ■
■ Multi-region       ■ Manual       ■ Built-in     ■ Manual       ■ Manual       ■
■ WebSocket support  ■ ■ Yes       ■ ■ Yes       ■ ■ Yes       ■ ■ No       ■
■ Streaming response ■ ■ Yes       ■ ■ Yes       ■ ■ Yes       ■ ■ No       ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
```

■ Real Exam Scenarios

Scenario 1: "You need to deploy a containerized API that handles 10K requests/day with 95% coming during business hours. Cost is a primary concern."

- **Answer:** Cloud Run (scales to zero at night, pay-per-request)
- **Why not Functions:** Container requirement
- **Why not App Engine:** Can't scale to zero (standard)
- **Why not GKE:** Overkill, always-on costs

Scenario 2: "Legacy Java app needs migration with minimal changes. Requires background task queues and scheduled jobs."

- **Answer:** App Engine Standard (built-in Task Queues, Cron)
- **Why not Cloud Run:** Need to refactor for task queues
- **Why not GKE:** Over-engineering for simple app

Scenario 3: "Microservices with service mesh, Istio, and complex networking."

- **Answer:** GKE (full Kubernetes control)
- **Why not Cloud Run:** Limited networking control

Scenario 4: "Process uploaded images: resize, watermark, save to bucket."

- **Answer:** Cloud Functions (event-driven, triggered by Cloud Storage)

- **Why not Cloud Run:** Overkill for simple transformation

■ Pub/Sub Patterns Deep Dive

At-Least-Once Delivery:

- Messages may be delivered multiple times
- **Must implement idempotency:**
- Use unique message IDs to track processed messages
- Store message IDs in Firestore/Memorystore with TTL
- Example: payment processing (don't charge twice!)

```
# Idempotent handler pattern
import redis
redis_client = redis.Redis()

def process_message(message_id, data):
    # Check if already processed (TTL 7 days)
    if redis_client.exists(f"processed:{message_id}"):
        return {"status": "already_processed"}

    # Process the message
    result = do_business_logic(data)

    # Mark as processed
    redis_client.setex(f"processed:{message_id}", 604800, "1")
    return result
```

Ordering Keys:

- Use when message order matters for a specific entity
- Example: user profile updates (create → update → delete)
- All messages with same ordering key go to same partition

```
gcloud pubsub topics create orders --message-ordering
gcloud pubsub subscriptions create orders-sub \
  --topic=orders --enable-message-ordering
```

Dead Letter Queues (DLQ):

- Poison messages that repeatedly fail processing
- After max attempts (5-100), move to DLQ for manual inspection
- Set up alerts when DLQ receives messages

■■ Database Selection Matrix

Use Case	Cloud SQL	Firestore	Spanner	Memorystore
Relational/SQL	■ Postgres	■ NoSQL	■ Yes	■ Cache
Global scale	■ Regional	■ Multi-reg	■ Global	■ Regional
Strong consist.	■ Yes	■■ Optional	■ Yes	■ Yes
Transactions	■ Full	■■ Limited	■ Full	■■ Limited
Max DB size	64 TB	Unlimited	Unlimited	300 GB
Read replicas	■ Yes	N/A	N/A	■ Yes
Automatic shard	■ No	■ Yes	■ Yes	■ No
Cost (relative)	$ Low	$$ Medium	$$$ High	$ Low
Best for	Legacy apps	Mobile/web	Finance/ERP	Session/cache

Key Decision Factors:

- **Cloud SQL:** Existing SQL app, < 10TB, single region, need read replicas
- **Firestore:** Mobile/web apps, flexible schema, multi-region reads, document model
- **Spanner:** Financial transactions, global consistency, unlimited scale, > 2TB
- **Memorystore:** Caching, sessions, pub/sub (Redis), sub-millisecond latency

■■ Common Exam Traps

1. **Trap:** "Use GKE for everything because it's most flexible"

- **Reality:** Over-engineering, higher cost, more complexity
- **Look for:** Simplest solution that meets requirements

2. **Trap:** "Firestore for all NoSQL needs"

- **Reality:** Consider Bigtable for time-series, analytics workloads

3. **Trap:** "Always use multi-region for HA"

- **Reality:** Regional can be sufficient (3 zones), lower latency/cost

Hands-on:

- Deploy a sample container to Cloud Run with min instances=1, concurrency=80, CPU throttling
- Set up Pub/Sub → Cloud Run push subscription with idempotent handler (Redis tracking)
- Create multi-region Cloud SQL instance with read replicas; test failover
- Implement circuit breaker pattern (exponential backoff, max retries)

Quick check:

- When to choose Spanner vs Cloud SQL? (Global scale vs regional, cost trade-off)

- How to guarantee idempotency with Pub/Sub? (Message ID tracking, Redis/Firestore)
- What's the difference between Cloud Run min-instances and App Engine min-instances? (Billing model)

Domain 2 — Build and test applications (High)

Topics and objectives:

- Build systems: Cloud Build triggers, build steps, private pools, caching; `cloudbuild.yaml`
- Artifact management: Artifact Registry (Docker, Maven, npm), vulnerability scanning
- Testing strategies: unit/integration/e2e, contract tests, mocks; Cloud Build + test reports
- Local dev: Cloud Code, Skaffold, Cloud Run emulator, Functions Framework
- Secure coding: input validation, least privilege, service accounts per service, Secrets management

■ Testing Pyramid Strategy

```
           /\              E2E Tests (Slow, Expensive)
          /  \             - Full user flows
         / E2E\            - Selenium, Puppeteer
        /______\           - Run on staging
       /        \
      /  Integr. \         Integration Tests (Medium)
     /   Tests    \        - API contracts
    /_____________\       - Database interactions
   /               \      - External service mocks
  /   Unit Tests    \     Unit Tests (Fast, Cheap)
 /                  \    - Pure functions
/____________________\   - 80% code coverage target
```

■■ Optimized Cloud Build Pipeline

```yaml
# cloudbuild.yaml - Production-ready example
options:
  machineType: 'E2_HIGHCPU_8'  # Faster builds
  substitutionOption: 'ALLOW_LOOSE'
  logging: CLOUD_LOGGING_ONLY

substitutions:
  _REGION: 'us-central1'
  _ARTIFACT_REGISTRY: '${_REGION}-docker.pkg.dev/${PROJECT_ID}/app-repo'

steps:
# Step 1: Restore cache (speeds up npm/pip installs)
- name: 'gcr.io/cloud-builders/gsutil'
  args: ['cp', 'gs://${PROJECT_ID}_cloudbuild/cache/node_modules.tar.gz', '.']
  id: 'restore-cache'

# Step 2: Install dependencies
- name: 'node:18'
  entrypoint: 'bash'
  args:
  - '-c'
  - |
    if [ -f node_modules.tar.gz ]; then
      tar -xzf node_modules.tar.gz
    fi
    npm ci
  id: 'install-deps'
  waitFor: ['restore-cache']

# Step 3: Run linting (parallel with tests)
- name: 'node:18'
  entrypoint: 'npm'
  args: ['run', 'lint']
  id: 'lint'
  waitFor: ['install-deps']

# Step 4: Run unit tests (parallel)
- name: 'node:18'
  entrypoint: 'npm'
  args: ['run', 'test:unit']
  env:
  - 'CI=true'
  id: 'unit-tests'
  waitFor: ['install-deps']

# Step 5: Run security scan (SAST)
- name: 'aquasec/trivy'
  args: ['fs', '--severity', 'CRITICAL,HIGH', '--exit-code', '1', '.']
  id: 'security-scan'
  waitFor: ['install-deps']

# Step 6: Build Docker image (wait for all tests)
- name: 'gcr.io/cloud-builders/docker'
  args:
  - 'build'
  - '-t'
  - '${_ARTIFACT_REGISTRY}/app:${SHORT_SHA}'
  - '-t'
  - '${_ARTIFACT_REGISTRY}/app:latest'
  - '--cache-from'
  - '${_ARTIFACT_REGISTRY}/app:latest'
  - '.'
  id: 'build-image'
  waitFor: ['lint', 'unit-tests', 'security-scan']
```

```
# Step 7: Scan image for vulnerabilities
- name: 'gcr.io/cloud-builders/gcloud'
  entrypoint: 'bash'
  args:
  - '-c'
  - |
    gcloud artifacts docker images scan ${_ARTIFACT_REGISTRY}/app:${SHORT_SHA} \
      --location=${_REGION} --format=json
  id: 'image-scan'
  waitFor: ['build-image']

# Step 8: Push image
- name: 'gcr.io/cloud-builders/docker'
  args: ['push', '--all-tags', '${_ARTIFACT_REGISTRY}/app']
  id: 'push-image'
  waitFor: ['image-scan']

# Step 9: Create provenance attestation (supply chain security)
- name: 'gcr.io/cloud-builders/gcloud'
  entrypoint: 'bash'
  args:
  - '-c'
  - |
    gcloud beta container binauthz attestations sign-and-create \
      --artifact-url=${_ARTIFACT_REGISTRY}/app:${SHORT_SHA} \
      --attestor=prod-attestor \
      --attestor-project=${PROJECT_ID}
  id: 'attestation'
  waitFor: ['push-image']

# Step 10: Update cache for next build
- name: 'gcr.io/cloud-builders/gsutil'
  entrypoint: 'bash'
  args:
  - '-c'
  - |
    tar -czf node_modules.tar.gz node_modules
    gsutil cp node_modules.tar.gz gs://${PROJECT_ID}_cloudbuild/cache/
  id: 'save-cache'
  waitFor: ['push-image']

images:
- '${_ARTIFACT_REGISTRY}/app:${SHORT_SHA}'
- '${_ARTIFACT_REGISTRY}/app:latest'

timeout: '1200s'  # 20 minutes
```

■ Secret Management Best Practices

■ Bad (Never do this):

```
env:
- DB_PASSWORD=mypassword123  # Exposed in logs/configs
```

■ Good (Secret Manager + Workload Identity):

```
steps:
- name: 'gcr.io/cloud-builders/gcloud'
  entrypoint: 'bash'
  args:
  - '-c'
  - |
    DB_PASSWORD=$(gcloud secrets versions access latest --secret=db-password)
    # Use $DB_PASSWORD in build steps
  secretEnv: ['DB_PASSWORD']

availableSecrets:
  secretManager:
  - versionName: projects/${PROJECT_ID}/secrets/db-password/versions/latest
    env: 'DB_PASSWORD'
```

Cloud Run with Secret Manager:

```
gcloud run deploy app \
  --image=${IMAGE} \
  --update-secrets=DB_PASSWORD=db-password:latest \
  --service-account=app-sa@${PROJECT_ID}.iam.gserviceaccount.com
```

■ Vulnerability Scanning Matrix

Severity	Action	Who Gets Notified
CRITICAL	■ Block deployment	Security + Dev lead
HIGH	■■ Review required	Dev lead
MEDIUM	■ Allow + ticket	Dev team
LOW	■ Allow (log)	Automated report

■ Private Pool Use Cases

1. **Access to VPC resources** (Cloud SQL, internal APIs)
2. **Larger machines** (custom machine types)
3. **Faster builds** (no cold start, persistent cache)
4. **Compliance** (data residency, private network)

```
# Create private pool in VPC
gcloud builds worker-pools create my-pool \
  --region=us-central1 \
  --peered-network=projects/${PROJECT_ID}/global/networks/default \
  --worker-machine-type=e2-standard-4

# Use in cloudbuild.yaml
options:
  pool:
    name: 'projects/${PROJECT_ID}/locations/us-central1/workerPools/my-pool'
```

Build Performance Optimization

Technique	Time Saved	Complexity
Layer caching	30-50%	Low
Multi-stage builds	20-40%	Low
Parallel steps	20-60%	Medium
Private pool	10-20%	Medium
Dependency caching	40-70%	Medium
Larger machine type	30-50%	Low

Hands-on:

- Create optimized `cloudbuild.yaml` with parallel test execution and caching
- Enable Artifact Registry vulnerability scanning with severity policies
- Configure Binary Authorization with attestation requirement
- Use Secret Manager with per-environment secrets (dev/staging/prod)
- Set up private worker pool with VPC access to Cloud SQL

Quick check:

- GitHub App vs Cloud Source Repositories? (GitHub App = full CI/CD, CSR = simple Git)
- Purpose of private pool? (VPC access, custom machine, compliance)
- How to pass secrets safely? (Secret Manager, not env variables)
- What's the difference between `waitFor: ['-']` and omitting it? (Parallel vs sequential)

Domain 3 — Deploy applications (High)

Topics and objectives:

- Progressive delivery: blue/green, canary, traffic splitting (Cloud Run, App Engine), rollbacks
- Infra as Code: Terraform basics for app infra; Cloud Deploy for release orchestration
- Config and environment management: Config Controller, ConfigMaps/Secrets (K8s), service-to-service auth
- Zero-downtime migrations and DB schema versioning (Liquibase/Flyway)

Hands-on:

- Deploy two revisions in Cloud Run and split traffic (90/10 → 50/50 → 100/0)
- Define a Cloud Deploy pipeline to promote artifacts from dev → staging → prod
- Practice DB migration with a reversible change and rollback plan

Quick check:

- How would you implement a canary in Cloud Run and auto-rollback on errors?

Domain 4 — Integrate applications with Google Cloud services (Medium–High)

Topics and objectives:

- API exposure: API Gateway vs Cloud Endpoints; auth with JWTs/OIDC; rate limiting
- Messaging & streaming: Pub/Sub features (DLQs, ordering), Dataflow templates (overview)
- Data services: Cloud SQL connectivity (private IP/Serverless VPC Access), Firestore indexes, Spanner schemas
- Storage: Cloud Storage signed URLs, object lifecycle, event notifications
- Identity and access: IAM roles, custom roles, Workload Identity Federation, per-service identities

Hands-on:

- Protect a Cloud Run service behind API Gateway with JWT/Authenticator
- Use Serverless VPC Access to reach Cloud SQL from Cloud Run
- Configure Pub/Sub DLQ and test poison message handling

Quick check:

- When to choose API Gateway vs Endpoints?
- How to set least-privilege for a build/deploy pipeline?

Domain 5 — Secure applications (High)

Topics and objectives:

- IAM best practices: least privilege, per-service accounts, minimal token scope, keyless auth
- Secret Manager vs env variables; KMS basics; CMEK for data services
- Organization policy constraints; Binary Authorization (GKE), Cloud Run invoker roles
- Supply chain security: Artifact signing (Sigstore), SBOMs, scanning

Hands-on:

- Replace env-based secrets with Secret Manager + per-revision access
- Enable Artifact Registry vulnerability scanning; block deploy on critical findings
- Configure Binary Authorization policy for GKE (if applicable)

Quick check:

- How do you rotate secrets without downtime?

Domain 6 — Monitor, log, and trace (High)

Topics and objectives:

- Observability stack: Cloud Logging, Cloud Monitoring, Error Reporting, Trace, Profiler, Debugger
- SLI/SLO/SLA basics; alerting policies; uptime checks; log-based metrics
- Distributed tracing for microservices; correlation IDs; structured logging
- Cost monitoring: quotas, budgets, labels

Hands-on:

- Emit structured JSON logs; create log-based metrics and alerts on error ratio
- Add OpenTelemetry auto-instrumentation to a service and view traces

- Define SLO (availability, latency) and attach alerting burn-rate policy

Quick check:

- What’s a burn-rate alert and why is it used?

High-Frequency Topic Map (What appears most)

- Cloud Run vs GKE vs App Engine decision making — High
- Cloud Build, Artifact Registry, CI test automation — High
- Traffic splitting, canary/rollback, Cloud Deploy — High
- Pub/Sub patterns: idempotency, DLQ, ordering — High
- Cloud SQL/Firestore/Spanner selection and connectivity — High
- IAM, Secret Manager, secure service-to-service auth — High
- Observability: logs, metrics, traces, SLOs, alerts — High

30-Day Study Plan (2 hrs/day)

- Week 1: Workload placement, resiliency, Pub/Sub patterns; Hands-on Cloud Run + Pub/Sub
- Week 2: CI/CD with Cloud Build, Artifact Registry; Security with Secret Manager; Canary deploys
- Week 3: Data services (SQL/Firestore/Spanner), VPC access, API Gateway; Observability stack
- Week 4: End-to-end project, SLOs + burn-rate alerts; Review sample questions and weak areas

Commands & Snippets

- Cloud Build trigger:

```
steps:
- name: 'gcr.io/cloud-builders/docker'
  args: ['build','-t','$REGION-docker.pkg.dev/$PROJECT/repo/app:$COMMIT_SHA','.']
- name: 'gcr.io/cloud-builders/docker'
  args: ['push','$REGION-docker.pkg.dev/$PROJECT/repo/app:$COMMIT_SHA']
images:
- '$REGION-docker.pkg.dev/$PROJECT/repo/app:$COMMIT_SHA'
```

- Cloud Run deploy with traffic split:

```
gcloud run deploy app \
  --image=$REGION-docker.pkg.dev/$PROJECT/repo/app:$COMMIT_SHA \
  --region=$REGION --platform=managed --allow-unauthenticated
# then split
gcloud run services update-traffic app \
  --to-revisions REV1=90,REV2=10 --region=$REGION
```

- Pub/Sub DLQ policy example:

```
gcloud pubsub subscriptions create sub \
  --topic=topic --dead-letter-topic=dlq --max-delivery-attempts=5
```

■

■ Cost Optimization Examples (Exam Favorite)

Scenario-Based Cost Analysis

Scenario 1: API Service (10K requests/day, 50ms avg)

```
Cloud Run (scale-to-zero):
- Requests: 10K/day = 300K/month
- vCPU: 1 vCPU @ $0.00002400/vCPU-sec
- Memory: 512MB @ $0.00000250/GB-sec
- Request: $0.40/million
- Monthly: ~$8-12

App Engine Standard:
- Instance hours: ~720 hrs (always 1 instance)
- F1 instance: ~$0.05/hr
- Monthly: ~$36

GKE:
- Node: e2-medium (2vCPU, 4GB) = $24.27/mo
- Cluster fee: $73/mo
- Monthly: ~$97

■ Winner: Cloud Run (saves 70-90%)
```

Scenario 2: Background Job (1 hour/day, CPU-intensive)

```
Cloud Run Jobs:
- 1 vCPU, 2GB, 1 hour/day
- Cost: ~$3/month

GKE CronJob:
- Always-on node even if job runs 1hr
- Cost: ~$97/month (node + cluster)

■ Winner: Cloud Run Jobs (saves 95%)
```

Scenario 3: Stateful Application (24/7, persistent storage)

```
GKE with StatefulSet:
- Right choice for stateful workloads
- Persistent volumes
- Cost: $97/mo + storage

Cloud Run:
- ■ Cannot persist state between requests
- Wrong choice

■ Winner: GKE (only option)
```

■

■ Common Anti-Patterns (What NOT to Do)

1. **Anti-pattern:** Storing secrets in environment variables

- **Problem:** Visible in logs, console, metadata
- **Solution:** Secret Manager with IAM

2. **Anti-pattern:** Using default service account

- **Problem:** Over-privileged (Editor role)
- **Solution:** Custom service account per service

3. **Anti-pattern:** Polling Pub/Sub in a loop

- **Problem:** Wastes CPU, increases latency
- **Solution:** Push subscription to Cloud Run

4. **Anti-pattern:** Building images on local machine

- **Problem:** "Works on my machine", no security scanning
- **Solution:** Cloud Build with automated scanning

5. **Anti-pattern:** Hard-coding configuration

- **Problem:** Different values per environment
- **Solution:** Environment variables or Secret Manager

6. **Anti-pattern:** No retry logic for external calls

- **Problem:** Transient failures break application
- **Solution:** Exponential backoff with max retries

■

■ Real-World Troubleshooting Scenarios

Problem 1: Cloud Run 503 errors under load

Symptoms:

- Works fine with low traffic
- 503 "Service Unavailable" at 100+ req/sec
- CPU not maxed out

Diagnosis:

```
# Check current concurrency setting
gcloud run services describe app --region=us-central1 \
  --format='value(spec.template.spec.containerConcurrency)'
```

Root cause: Default concurrency=80, but your app is single-threaded

Solution:

```
# Reduce concurrency for single-threaded apps
gcloud run services update app --concurrency=1 --region=us-central1
# OR make app multi-threaded
```

Problem 2: Cloud Build timeout after 10 minutes

Symptoms:

- Build fails with "Build timeout"
- npm install takes 8 minutes
- Tests take 5 minutes

Diagnosis: Default timeout is 10 minutes

Solutions:

```
# Option 1: Increase timeout
timeout: '3600s'  # 1 hour

# Option 2: Parallel steps (better)
steps:
- name: 'node'
  args: ['npm', 'install']
  id: 'install'
- name: 'node'
  args: ['npm', 'run', 'test:unit']
  waitFor: ['install']  # Parallel after install
  id: 'test-unit'
- name: 'node'
  args: ['npm', 'run', 'test:integration']
  waitFor: ['install']  # Parallel
  id: 'test-int'

# Option 3: Caching (best)
# Cache node_modules between builds
```

Problem 3: Pub/Sub messages processed multiple times

Symptoms:

- Payment charged twice
- Database records duplicated
- Message count shows delivery but backlog increases

Root cause: Not idempotent, acknowledgement failures

Solution:

```
import hashlib
from google.cloud import firestore

db = firestore.Client()

def process_message(message):
    message_id = message.message_id
    doc_ref = db.collection('processed').document(message_id)

    # Atomic check-and-set
    transaction = db.transaction()
    snapshot = doc_ref.get(transaction=transaction)

    if snapshot.exists:
        return {'status': 'duplicate', 'message_id': message_id}

    # Process message
    result = do_business_logic(message.data)

    # Mark as processed (atomic)
    doc_ref.set({'processed_at': firestore.SERVER_TIMESTAMP})

    return result
```

Problem 4: Cloud SQL connection exhaustion

Symptoms:

- "Too many connections" error
- Cloud Run scales up, database fails
- Max connections: 100

Root cause: Each Cloud Run instance creates connection pool

Solutions:

1. **Connection pooling:**

```
# Use Cloud SQL Python Connector with connection pooling
from google.cloud.sql.connector import Connector
import sqlalchemy

connector = Connector()

def getconn():
    return connector.connect(
        "project:region:instance",
        "pg8000",
        pool_size=5,  # Limit per instance
        max_overflow=2
    )

pool = sqlalchemy.create_engine(
    "postgresql+pg8000://",
    creator=getconn,
    pool_size=5,  # Max 5 connections per instance
    max_overflow=2
)
```

2. **Set max-instances:**

```
# Limit Cloud Run instances
gcloud run services update app \
  --max-instances=15 \
  --region=us-central1
# 15 instances × 5 conn = 75 connections (< 100 limit)
```

3. **Use Cloud SQL Proxy:**

- Handles connection pooling automatically
- Encrypted connections

■

■ Exam Strategy & Tips

Reading Questions

1. **Highlight key constraints:**

- Budget concerns → Prefer serverless
- Compliance/regulations → Organization policies, CMEK
- "Minimal changes" → Lift-and-shift (App Engine, GKE)
- "Cloud-native" → Cloud Run, Functions

- "Existing K8s" → GKE

2. **Watch for keywords:**

- "Cost-effective" = Cloud Run > App Engine > GKE
- "Low latency" = Set min-instances, use Memorystore
- "Secure" = Secret Manager, private endpoints, least privilege
- "Scalable" = Autoscaling, stateless, managed services
- "Reliable" = Multi-region, retry logic, error budgets

Decision Framework

```
Question: How to deploy X?

1. What's the constraint?
   - Cost → Serverless
   - Control → GKE
   - Simplicity → Cloud Run
   - Legacy → App Engine

2. What's the workload?
   - HTTP API → Cloud Run
   - Events → Functions
   - Batch → Cloud Run Jobs
   - Stateful → GKE

3. What's the scale?
   - Variable → Autoscaling (Run/Functions)
   - Steady → App Engine
   - Huge → GKE + HPA
```

Time Management

- 2 hours for 50-60 questions = ~2 minutes per question
- **Strategy:**
- First pass: Answer easy questions (30-40 min)
- Mark difficult ones for review
- Second pass: Tackle marked questions (40-50 min)
- Final review: Check marked answers (20-30 min)

Red Flags (Wrong Answers)

- ■ Over-engineering ("Use GKE for simple API")

- ■ Ignoring managed services ("Build your own load balancer")
- ■ Security holes ("Store keys in code")
- ■ No error handling ("Just deploy and hope")
- ■ Ignoring cost ("Use biggest machines")

Final Review Checklist

- [ ] Clear on workload placement trade-offs
- [ ] Can build/test artifacts and enforce scanning
- [ ] Can deploy progressively and rollback safely
- [ ] Can integrate with managed data services securely
- [ ] Emits structured logs and traces; SLOs with alerts
- [ ] Familiar with sample questions and timing