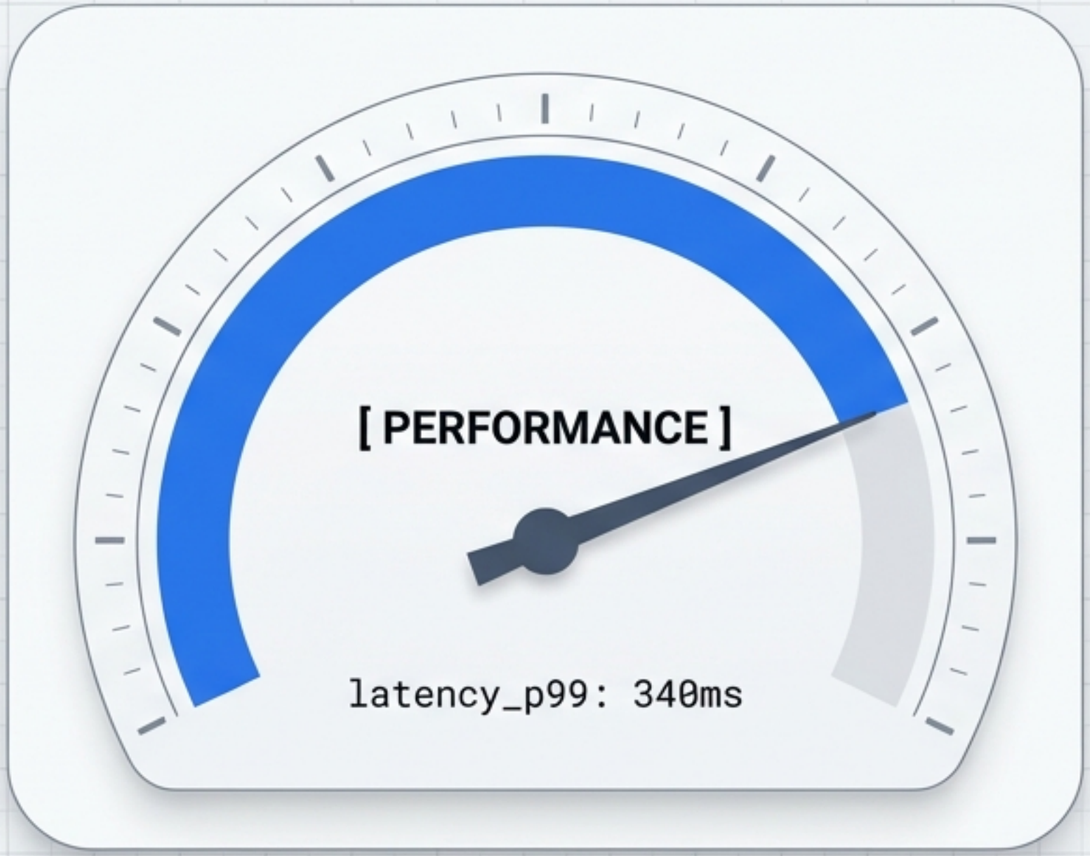
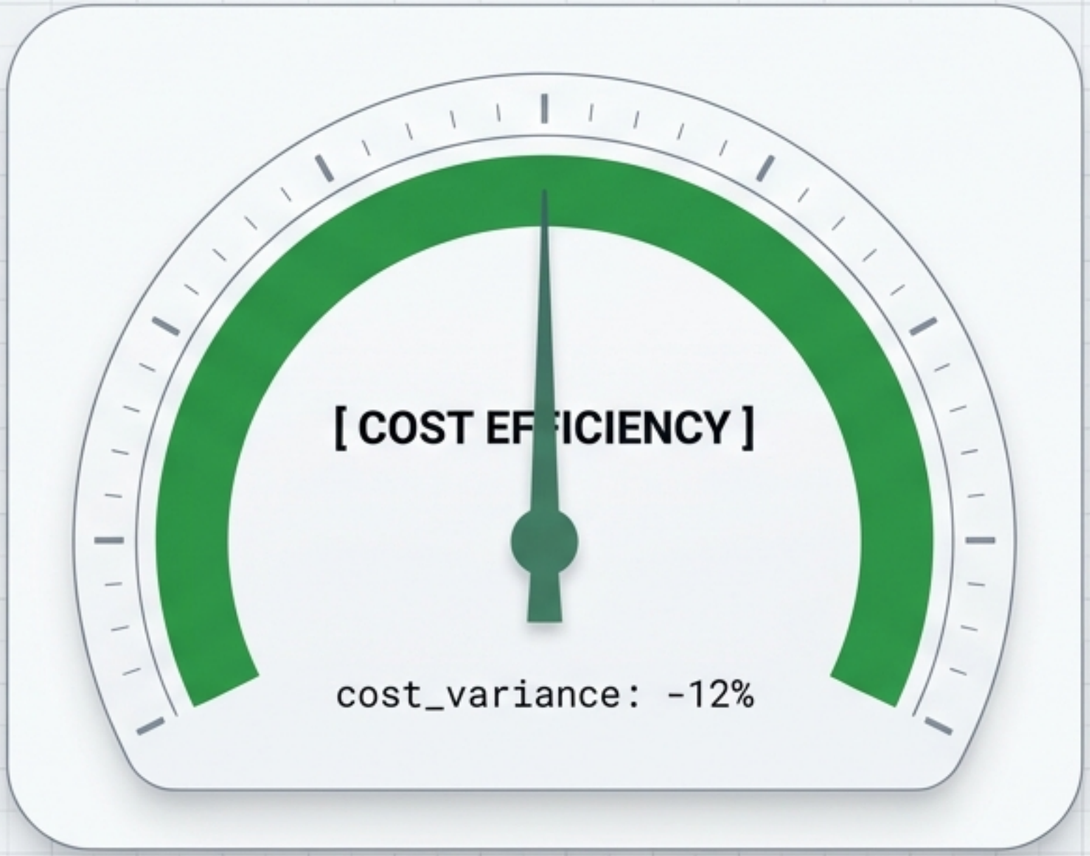
PDE Section 5: The Optimization Blueprint

Mastering Performance and FinOps for Google Cloud DevOps





The Section 5 Operator's Console

Section 5.1: Performance Diagnostics

Compute

Cloud Run Execution Environments & CPU Allocation

Observability

Cloud Profiler, Cloud Trace, Cloud Monitoring

GKE

Vertical Pod Autoscaling & Rightsizing

Section 5.2: FinOps & Cost Optimization

Compute

Scale-to-Zero, Concurrency, Spot VMs

Analytics

BigQuery Export & Cost Breakdown Reports

Controls

CUDs, Recommender, Budget Alerts

Target: ~12% of the PDE Exam. Goal: Maximize workload density, minimize latency, and eliminate idle waste.

Cloud Run Execution Environments: Gen 1 vs. Gen 2

Gen 1 (Legacy Sandbox)

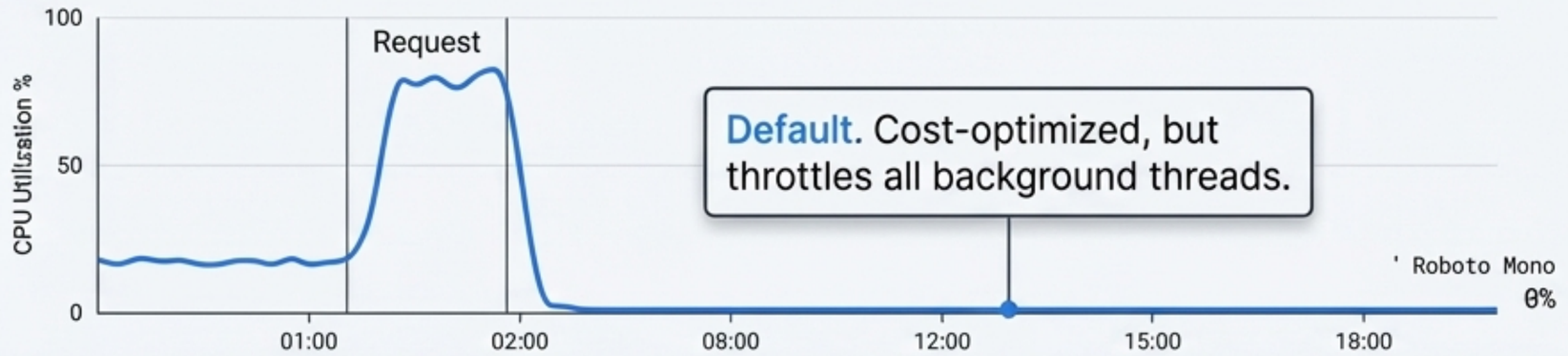
Architecture:	Sandbox isolation
Cost:	Lower baseline cost
Best For:	Simple, stateless workloads, highly scalable web APIs.
Capabilities:	✗ No Unix sockets
	✗ Single-threaded focus

Gen 2 (Full Linux Compatibility)

Architecture:	`gVisor` isolation (Full Linux)
Performance:	Improved CPU for compute-heavy; faster startup for memory-heavy.
Capabilities:	✓ Multi-threading support
	✓ Unix sockets
Recommendation:	Default for all new workloads unless strictly optimizing for absolute lowest baseline cost.

The Latency Cost of CPU Throttling

cpu_idle = true



cpu_idle = false

The OPcache Case Study

Throttled (Gen 1): OPcache generation competed with request serving. p99 latency spiked to 2.1 seconds.

Always-On (Gen 2): Continuous background warming allowed cold-start p99 to drop to 340 milliseconds with zero code changes.

The Observability Trinity: Diagnostic Decision Tree

What bottleneck are you hunting?

CPU/Memory hotspots in specific functions?

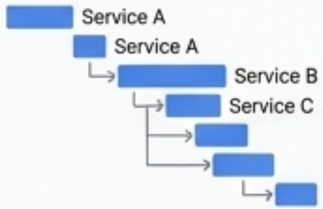
in Roboto Mono

Cloud Profiler

Continuous production profiling (<1% overhead). Identifies which function eats 60% of CPU.

End-to-end distributed latency?

in Roboto Mono

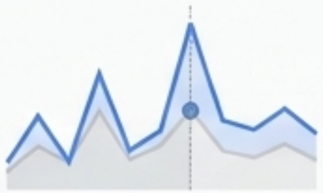


Cloud Trace

Tracks downstream bottlenecks across microservices via the `X-Cloud-Trace-Context` header.

Infrastructure saturation or tail latency?

in Roboto Mono

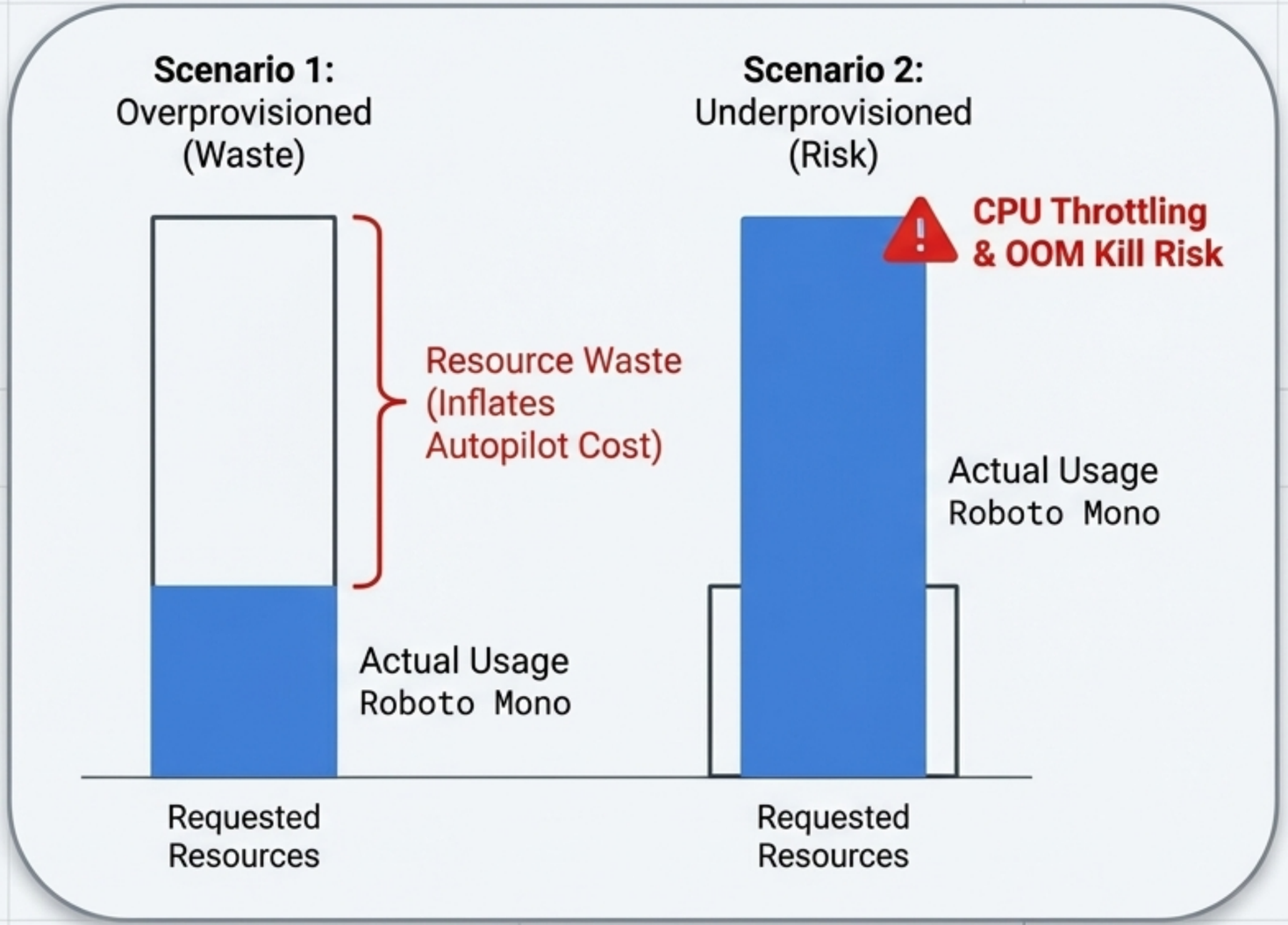


Cloud Monitoring

Use Metrics Explorer. Key metric: `run.googleapis.com/request_latencies` (Use `ALIGN_PERCENTILE_99`).

GKE Rightsizing and the Vertical Pod Autoscaler

GKE Rightsizing and the Vertical Pod Autoscaler



Closing the Gap

- **The Mechanism:** GKE Autopilot bills based on requested resources, not node size.
- **The Solution:** Vertical Pod Autoscaler (VPA). Analyzes actual usage over time to recommend exact CPU/memory requests.

  `kubectl get vpa -n [namespace]`

- **UI Path:** Kubernetes Engine > Workloads > `[deployment]` > Observability tab.

Entering FinOps: Restricting the Fuel

Objective 5.2: Implementing FinOps practices.

The Mission: Shift focus from raw performance to maximizing workload density. Eliminate idle compute and precisely size resources to minimize financial waste without starving the engine.

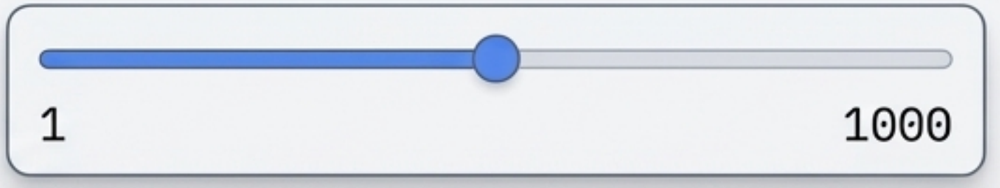
Cloud Run Cost Controls: Scale-to-Zero & Concurrency

Dial 1: Minimum Instances

Impact: Eliminates compute costs for idle workloads (60-90% savings for uneven traffic).

Trade-off: 1-3 second cold start for pre-built containers. (Setting to 1 prevents cold starts but incurs continuous base cost).

Dial 2: Concurrency



Rule of Thumb for Exam:

I/O-Bound (Waiting on DBs/APIs): Set HIGH (80-1000) to maximize instance sharing and reduce cost.

CPU-Bound (Image processing, ML): Set LOW (1-10) to prevent request contention and degraded latency.

GKE Cost Control: The Spot VM Advantage

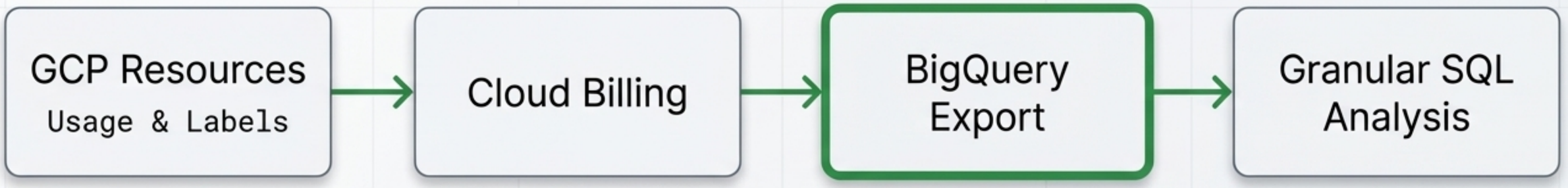
Autopilot Billing Truth

> You are billed for what you request, not what the underlying node provides. Precision in `resources.requests` is the primary cost driver.

Spot VM Lifecycle

Key Exam Facts	
Discount:	60-91% off standard VMs.
Lifespan:	NO maximum lifespan (unlike 24-hour limit on old Preemptible VMs). Runs until Google needs capacity.
Best Fit:	Fault-tolerant batch workloads, ML training, CI/CD agents.
Implementation:	Set `spot: true` in node pool spec. Use **Kubernetes tolerations/nodeSelectors** to isolate from latency-sensitive services.

The FinOps Analytics Engine



The Foundation

Exporting Cloud Billing data to BigQuery is the foundation of any GCP FinOps practice.

Enables SKU-level granularity.

The Goal

Showback and chargeback. Run SQL to determine exactly which Cloud Run service consumed the most CPU-seconds, or which namespace generated the most egress.

Cost Breakdown

Use Cost Breakdown Reports (Billing > Reports) to view net costs after Committed Use Discounts (CUDs) and Sustained Use Discounts (SUDs) are applied.

The **FinOps Toolchain:** Defense vs. Offense

Reactive (Defense) - Looking Backward

Tool:
`Budget Alerts`

Mechanism:
Scoped to specific projects/services. Triggers at set thresholds (e.g., 50%, 90%, 100%).

Action:
Connect to Pub/Sub topics to trigger automated responses (e.g., a Cloud Function that scales down non-prod services at 90% spend).

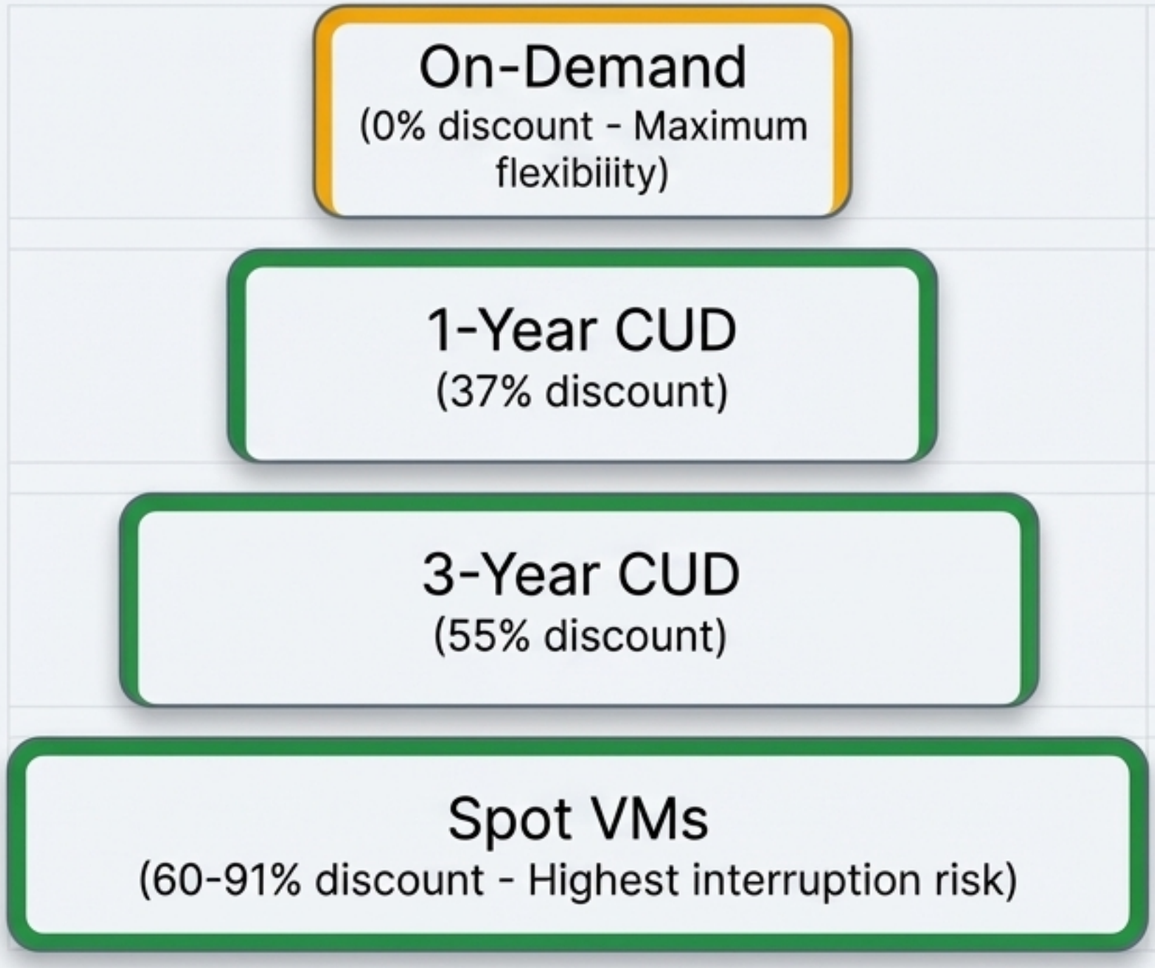
Proactive (Offense) - Looking Forward

Tool:
`Active Assist / Recommender`

Mechanism:
ML-driven analysis of 30 days of historical usage data.

Key Targets:
- Idle VMs (<5% average CPU for 14 days).
- VM rightsizing (downsizing over-provisioned Compute Engine).
- Overprovisioned Cloud Run CPU allocations.

GKE Committed Use Discounts (CUDs)

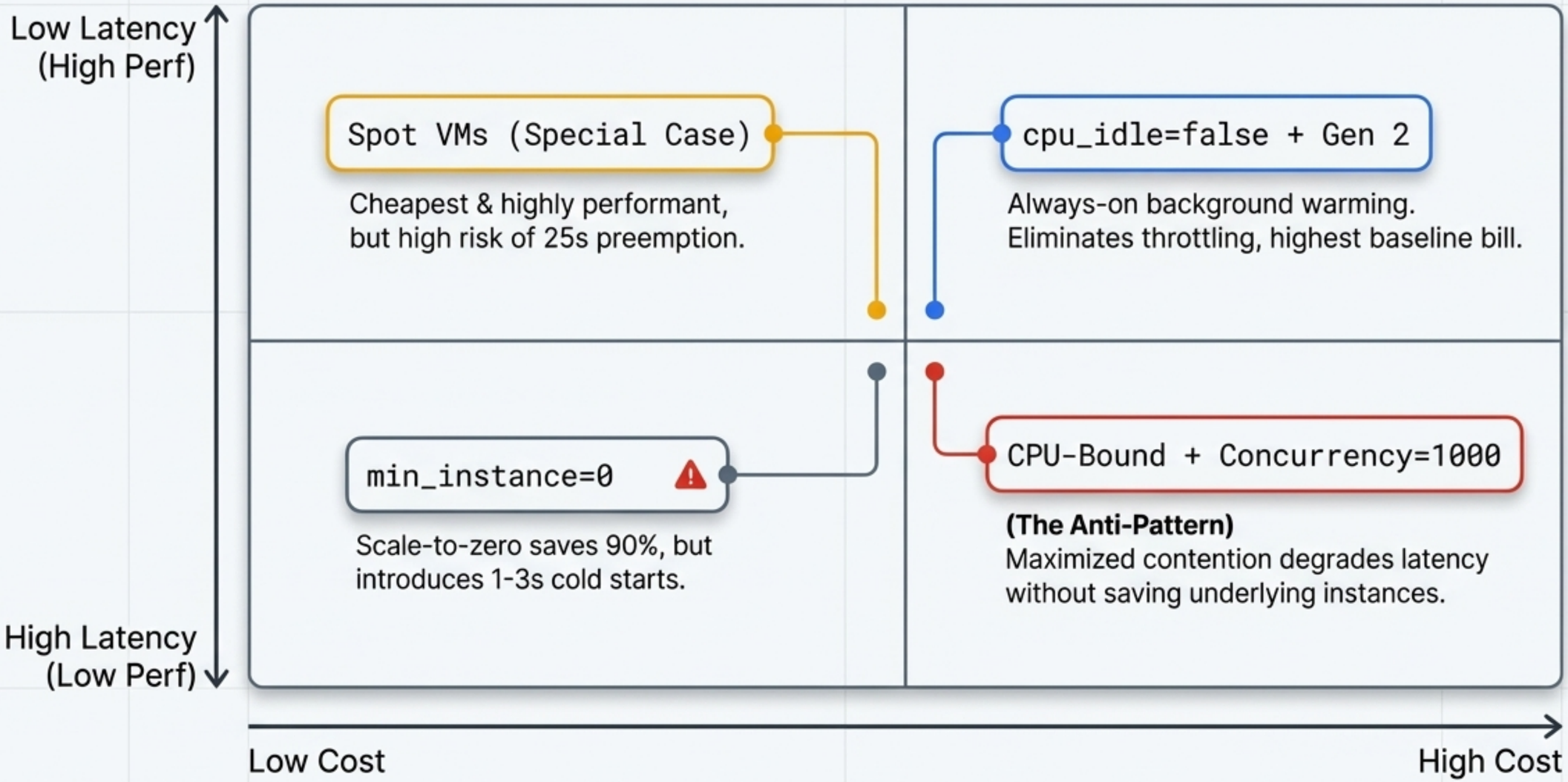


Crucial Exam Distinction

Standard Compute Engine CUDs:
You commit to specific machine types in specific geographic regions.

GKE Autopilot CUDs:
You commit to an aggregate resource amount (total `vCPU-hours` and `GB-hours`). This financial commitment can be consumed by any pod, across any namespace or workload, offering massive flexibility.

Synthesis: The Cost-Performance Balancing Act



Final Exam Checklist: Paths & Commands

```
Console Paths to Memorize:
> Billing > Billing export > BigQuery export
  // Foundation of granular FinOps

> Monitoring > Metrics Explorer > run.googleapis.com/request_latencies
  // Use ALIGN_PERCENTILE_99 for tail latency

> Kubernetes Engine > Workloads > [deployment] > Observability
  // Integrated GKE metrics without opening Monitoring

> Billing > Cost breakdown
  // View net cost after CUDs/SUDs

Commands:
> kubectl get vpa -n [namespace]
  // View VPA recommendations to close requested/actual gap
```

Optimize the engine. Restrict the fuel. Pass the exam.