DVA-C02 Pricing & Governance

Cost-aware developer choices and governance controls that show up on the exam.

BEST FOR	Reducing confusion between cost tools and control layers
FOCUS	Visibility, cost signals, security controls, and if/then rules
USE WITH	Quick Summary + Service Map

1. Cost visibility tools

Cost Explorer — Use for historical spend trends, usage analysis, and cost drill-down.
Budgets — Use when you need alerts against thresholds, not detailed optimization advice.
Cost allocation tags — Use to split spend by app, team, or environment.
CloudWatch metrics — Use for capacity and performance signals that drive right-sizing decisions.
CUR — Use when deep granular billing analysis is needed outside the console.

2. Governance controls for developers

IAM — Controls who can call which APIs.
KMS — Controls encryption keys and key usage.
Secrets Manager — Controls secret storage and rotation workflows.
CloudTrail — Records API activity for audit and investigation.
Config per environment — Use AppConfig, parameters, and IaC instead of hard-coded settings.

3. If / then rules

If traffic is spiky — then favor Lambda or autoscaling before fixed always-on capacity.
If reads dominate — then consider cache layers before scaling the primary database.
If deployments are risky — then use canary or blue/green with automated rollback signals.
If credentials appear in code — then move them to roles or Secrets Manager immediately.
If one queue is overloaded — then separate workloads by priority or consumption profile.

4. Developer-side cost traps

Overprovisioned Lambda memory — More memory can speed execution, but blind over-sizing wastes money.
Unbounded logging — Verbose logs can become a hidden cost multiplier.
Scan-heavy database access — Inefficient reads increase cost and hurt latency.
Idle environments — Old test stacks and orphaned artifacts keep billing silently.
Cross-service chatter — Chatty architectures can add network, request, and processing cost.

5. Optimization habits **Set retention** — Trim logs and artifacts with explicit lifecycle rules. **Use alarms** — Catch spikes before they become a surprise bill. **Tag consistently** — Without tags, shared environments become hard to manage. **Automate cleanup** — Destroy ephemeral test environments after validation. **Prefer managed fit** — Use the simplest managed service that matches the app pattern.	