AWS Certified Generative AI Developer - Professional Cheat Sheet

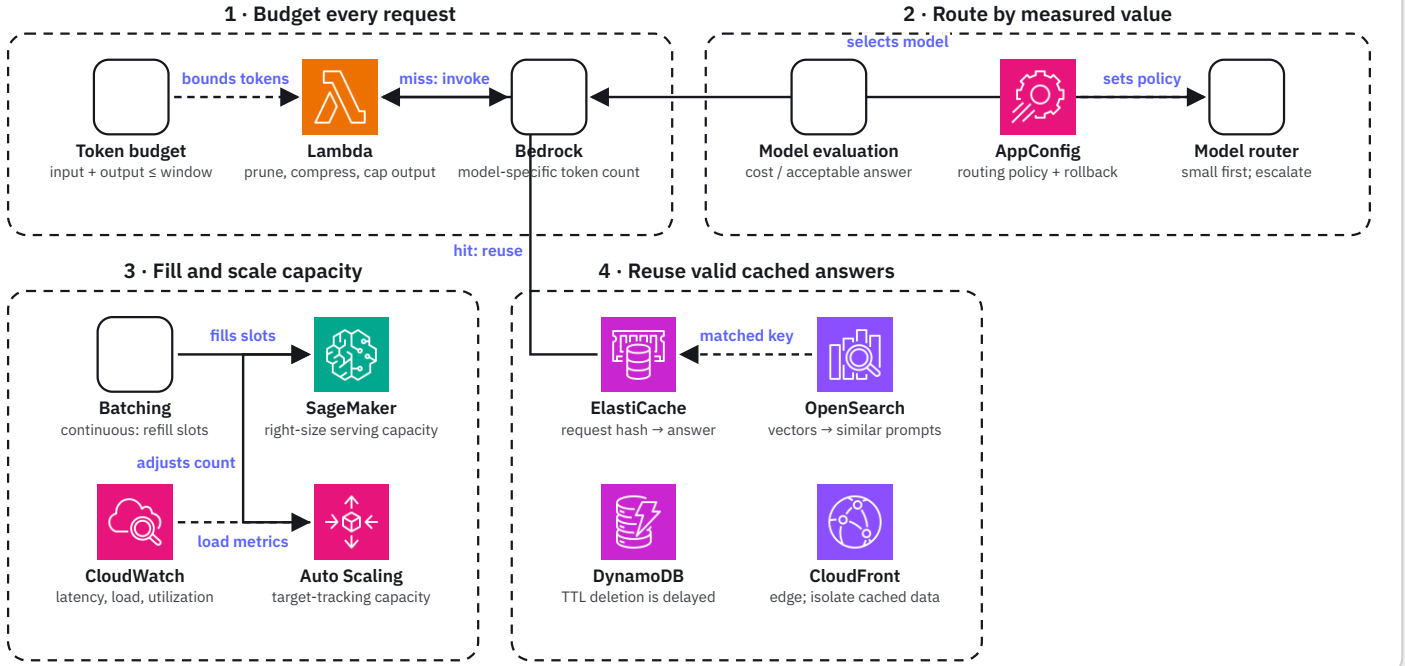
AIP-C01 · Domain 4 of 5

BlowStack

DOMAIN 4 · OPERATIONAL EFFICIENCY AND OPTIMIZATION FOR GENAI APPLICATIONS · 12% OF THE EXAM · TASK STATEMENT 4.1

Implement cost optimization and resource efficiency strategies

Budget, route, scale, cache



1 Budget every request

Use **token budgets** per model: prune irrelevant context, evaluate compression, and limit output with token caps and stop sequences. Count tokens, not words; reserve output space within the context window.

2 Route by measured value

Route by **quality and latency**, not token price alone; evaluate representative requests and total cost per acceptable answer. AWS AppConfig manages policy changes; validators and rollback alarms require configuration.

3 Fill and scale capacity

Use **batch inference** for delay-tolerant work; online inference serves live requests. Continuous batching refills active slots on supported stacks. Plan quotas, scaling lag, and minimum capacity before traffic peaks.

4 Reuse valid cached answers

Cache hits skip inference. Hash normalized request, model/settings, tenant, and context version for exact reuse. Semantic hits require similarity and authorization checks; reject expired entries and invalidate stale data.

Exam traps

- **Prompt caching is not response caching**: Bedrock reuses supported prompt-prefix computation, but still performs inference. A valid response-cache hit avoids the model invocation.
- **Provisioned throughput is not automatic elasticity**: size Bedrock commitments against sustained demand. AWS Auto Scaling does not automatically resize Bedrock provisioned throughput.
- **Max output tokens** limits generation, not input-token charges. A larger context window does not make long prompts cheaper; stop sequences can also truncate valid answers.

Get practice exams and lessons for AWS Certified Generative AI Developer - Professional

AWS Certified Generative AI Developer - Professional Cheat Sheet

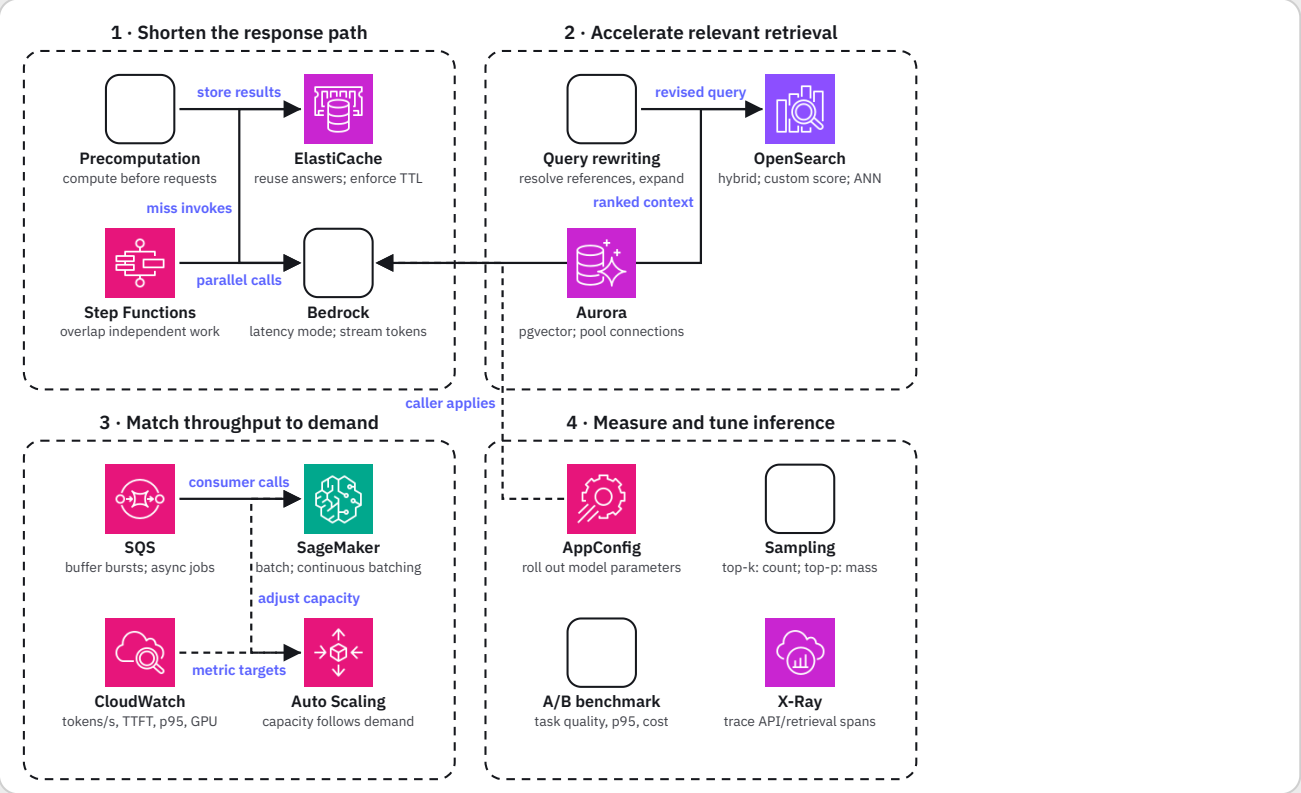
AIP-C01 · Domain 4 of 5

BlowStack

DOMAIN 4 · OPERATIONAL EFFICIENCY AND OPTIMIZATION FOR GENAI APPLICATIONS · TASK STATEMENT 4.2

Optimize application performance

Cache, retrieve, batch, tune



1 Shorten the response path

Predictable, repeated queries favor precomputation and caching with freshness controls. Parallel branches reduce sequential waits; Bedrock latency-optimized inference applies only to supported models and Regions.

2 Accelerate relevant retrieval

Hybrid search combines exact-term and semantic matches; scoring or reranking controls relevance. Tune vector indexes against recall and latency. Rewriting adds a model call; connection pooling avoids repeated setup.

3 Match throughput to demand

Token demand, not request count alone, drives capacity planning. Bound concurrency and token rates; queues absorb bursts, not increase quotas. Scale on measured utilization; right-size after profiling and allow warm-up.

4 Measure and tune inference

Task-specific wins require comparable A/B tests of quality, latency, and cost. Lower temperature reduces variability, not hallucinations. Parameter support varies by model; instrumented traces identify slow workflow spans.

Exam traps

- **Streaming** improves perceived responsiveness by exposing output early; it does not inherently reduce total generation time, computation, or token charges.
- **Batch inference** processes accumulated work offline in Bedrock or SageMaker. Continuous batching fills active serving batches as requests finish; it optimizes serving throughput.
- **Equal request counts** do not imply equal load: tokenization is model-specific, and prompt/output lengths vary. Shorter outputs reduce decoding work; overly tight limits can truncate answers.

Get practice exams and lessons for AWS Certified Generative AI Developer - Professional

AWS Certified Generative AI Developer - Professional Cheat Sheet

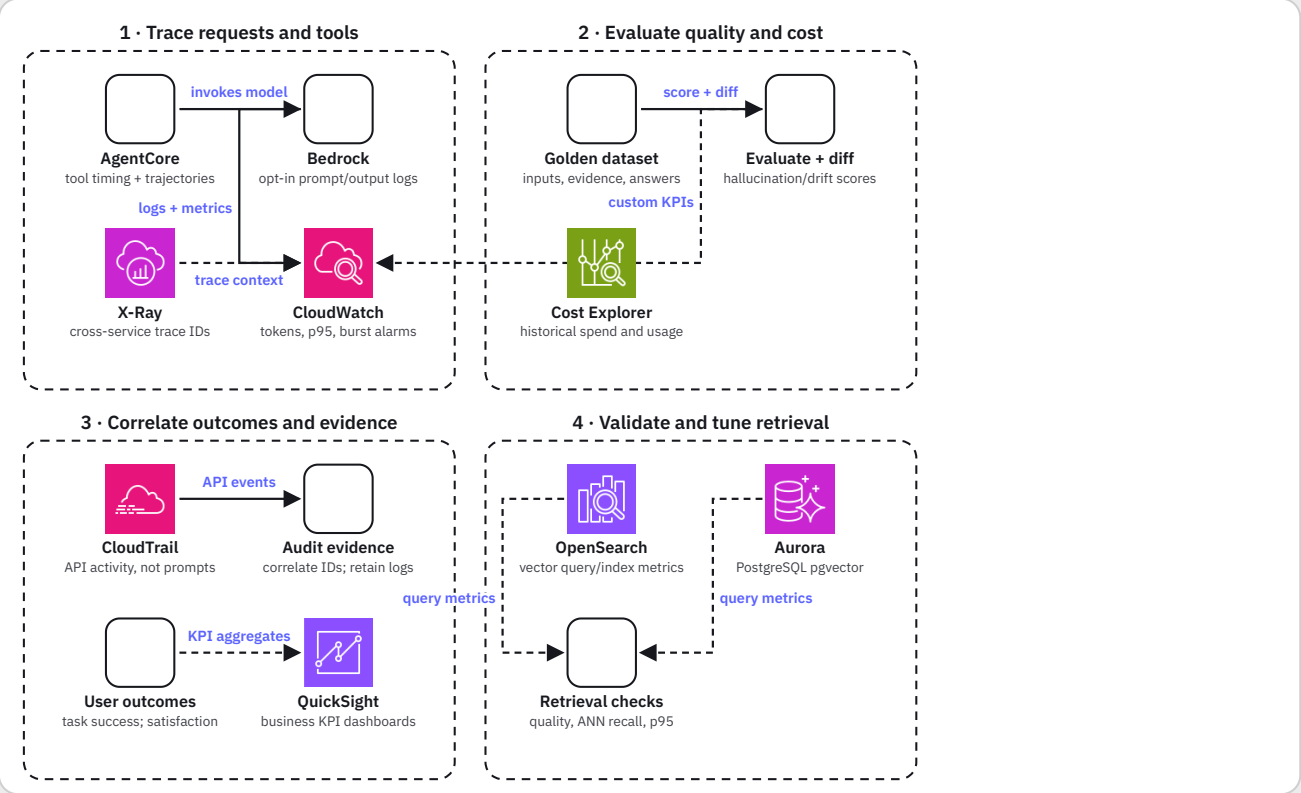
AIP-C01 · Domain 4 of 5

BlowStack

DOMAIN 4 · OPERATIONAL EFFICIENCY AND OPTIMIZATION FOR GENAI APPLICATIONS · TASK STATEMENT 4.3

Implement monitoring systems for GenAI applications

Trace, evaluate, correlate, tune



1 Trace requests and tools

Use **distributed traces** to localize model, retrieval and tool delays. Track tool errors, retries and agent handoffs against baselines; evaluate trajectories because a correct final answer can hide failed steps.

2 Evaluate quality and cost

Use **golden datasets** to benchmark prompt effectiveness and response quality. Repeat controlled runs to distinguish regressions from nondeterminism; publish custom quality metrics. Cost reports explain spend, not factual accuracy.

3 Correlate outcomes and evidence

Join request/session IDs across audits, invocation logs and user interactions. **Evidence integrity** requires access controls, tamper protection and retention policies; business KPIs require defined targets and review periods.

4 Validate and tune retrieval

Tune **recall versus latency** using representative queries. Validate freshness, embedding dimensions and missing vectors; schedule index maintenance explicitly. Monitoring alone does not repair data or optimize indexes.

Exam traps

- **API auditing ≠ interaction logging**: AWS CloudTrail does not replace Amazon Bedrock model invocation logs. Prompt/response capture requires explicitly enabled invocation logging.
- **Anomaly ≠ hallucination**: Amazon CloudWatch can flag token bursts or custom quality-score deviations; it does not automatically judge factual correctness or explain response drift.
- **ANN ≠ exact search**: faster retrieval can miss relevant neighbors. Low retrieval latency alone does not establish retrieval quality or grounded answers; measure recall against a benchmark.

Get practice exams and lessons for AWS Certified Generative AI Developer - Professional

AWS Certified Generative AI Developer - Professional Cheat Sheet

AIP-C01 · Domain 4 of 5

BlowStack

DOMAIN 4 · OPERATIONAL EFFICIENCY AND OPTIMIZATION FOR GENAI APPLICATIONS · WHAT TO REMEMBER

Concepts and services of Operational Efficiency and Optimization for GenAI Applications

Concepts to know

Token Budgeting

Allocates tokens across instructions, history, retrieval, tools, and output. Choose **pruning, compression, and output limits** to reduce cost and latency while preserving task success; measure actual usage with the model's tokenizer.

Inference Latency

Measures model response time; prompt length, output length, and capacity affect it. Choose **streaming for earlier visible output**, caching for reusable answers, and batch inference when immediate responses are unnecessary.

Model Evaluation

Compares models and application changes against representative tasks and quality, latency, and cost targets. Choose the **lowest cost per successful task**, not simply the cheapest tokens; use benchmark datasets to detect regressions.

Core services

Amazon Bedrock

Choose **managed foundation model inference** with model-dependent streaming, prompt caching, and batch options. Provisioned Throughput suits predictable demand when sustained utilization justifies cost.

Amazon CloudWatch

Choose **operational metrics, dashboards, and alarms** for token usage, latency, errors, and throttling. Response quality and hallucination rates require evaluation-derived custom metrics.

AWS Auto Scaling

Choose **demand-based scaling of supported resources** to reduce idle capacity and sustain throughput. Scaling application compute does not increase Amazon Bedrock invocation quotas.

Get practice exams and lessons for AWS Certified Generative AI Developer - Professional