Runbook: Quantum-Enhanced Log Clustering (Hybrid Quantum–Classical) – SRE Use Case

Last updated: 2026-01-23

Purpose

Provide a practical, repeatable procedure to prototype and operate a hybrid quantum–classical clustering pipeline for SRE log streams. The goal is to reduce triage time by grouping similar incidents and surfacing representative exemplars (centroids) for each cluster.

When to Use This Runbook

- You have large volumes of semi-structured logs (build logs, incident notes, app logs) with many near-duplicate patterns.
- Classical clustering struggles with high-dimensional embeddings or needs frequent tuning.
- You want an experiment-ready path to evaluate quantum optimization (QUBO/QAOA) for clustering or centroid selection.

Architecture Overview (High-level)

- Ingest: Kafka (or batch from S3) → preprocessor
- Embed: Text → embedding vectors (e.g., OpenAI/Azure embeddings)
- Candidate Reduction: Classical KMeans/HDBSCAN to form candidate groups (optional but recommended)
- Quantum Step: Formulate QUBO for assignment/centroid selection, solve with QAOA (or sampler)
- Store: Cluster labels + exemplars in SQLite/Postgres + vector store metadata
- Serve: RAG/LLM answers cite runbooks + exemplars; dashboard shows top clusters and drift

Inputs / Outputs

Item	Details
Input	Log messages (stream/batch), timestamps, service metadata
Intermediate	Embeddings, candidate clusters, QUBO matrix
Output	Cluster IDs, exemplars/centroids, confidence score, drift indicators

Pre-flight Checks

- Dataset sanity: confirm you have enough logs per service/time window; remove obvious noise (IDs, GUIDs) via normalization.
- Embedding consistency: use a single embedding model/version per environment; store model name in metadata.
- Dimensionality: if embeddings are large, consider PCA/UMAP for candidate reduction (do not reduce if you need full fidelity for similarity search).
- Quantum backend: confirm access to a simulator (local) and optionally hardware; set a max runtime budget.

Step-by-Step Procedure

- 1) Normalize logs: redact secrets, strip timestamps/IDs, standardize error codes.

- 2) Generate embeddings for each log line or event group.
- 3) (Optional) Candidate reduction: classical clustering to create small candidate sets per time window/service.
- 4) Build QUBO: encode objective to maximize intra-cluster similarity and minimize cross-cluster assignment; add constraints (one cluster per point).
- 5) Solve: run QAOA (or annealer/sampler) to find assignment/centroid decisions.
- 6) Post-process: compute silhouette-like score; label low-confidence items for manual review.
- 7) Persist: store cluster label + exemplar IDs; write summary to dashboard store.

QUBO / QAOA Practical Guidance

- Keep problem sizes small (e.g., 20–200 items) by windowing (time/service) + candidate reduction.
- Use stable IDs for log events to ensure reproducible results and incremental updates.
- Start on a simulator with a fixed seed; log parameters: p-level, shots, optimizer settings.
- Benchmark against a classical baseline (KMeans/HDBSCAN) using the same embeddings and scoring.

Operational Monitoring (SRE)

- Quality: cluster purity (manual sample), exemplar representativeness, % of unassigned/low-confidence items.
- Performance: embedding latency, quantum solve time, end-to-end pipeline SLA.
- Drift: rising number of new clusters over time, centroid movement, sudden similarity distribution shifts.

Failure Modes & Mitigations

- High noise → normalize better; group related log lines into an event.
- Empty/weak clusters → increase candidate set size or adjust similarity metric.
- Quantum runtime too high → reduce window size; lower QAOA depth; use better initialization from classical clustering.
- Non-deterministic results → fix seeds, log all hyperparameters, rerun baseline for comparison.

Sample Pseudocode (Conceptual)

```
logs = ingest()
clean = normalize(logs)
vecs = embed(clean)
candidates = classical_reduce(vecs) # optional
Q = build_qubo(candidates, similarity='cosine')
solution = solve_qaoa(Q, p=2, shots=1024)
labels, exemplars = decode(solution)
store(labels, exemplars); publish_dashboard(labels)
```

LLM Answering Guidance (RAG)

- When user asks a 'quantum question', retrieve this runbook plus the most similar exemplars from recent clusters.
- Require the LLM to cite: runbook section + exemplar IDs + time window.
- If retrieval score is low, answer with 'best effort' plus recommended diagnostics instead of guessing.