Model rollout to a serving fleet

front: what to ask, what to design · semi-guided, expect a twist · one problem

1 REQUIREMENTS

0–5 min · questions first, then sentences

- **Ask:** what is a worker: host, process or GPU? one loaded version per GPU? spare capacity for a rolling update?
- **Ask:** artifact size, immutable, version + checksum? origin outbound and per-worker link? fleet size and regions?
- **Ask:** must V1 keep serving during the rollout? what must be true before the first request reaches V2? what does "done" mean to the engineer?
- **A release engineer can:** publish a version · watch progress · pause, resume, roll back. **The system:** every targeted worker ends with a verified copy and serves it; the service stays available.
- **Technical challenges:** the origin link is the bottleneck · verify before serving · readiness = exact version · capacity-safe cohorts (`20%` headroom) · canary stop signals · resumable, idempotent, generation-checked reports.
- **Anchors:** `100 GB` artifact · `100 GB` GPU memory · `10 Gbit/s` origin and per-worker · 10 workers then `1,000` in 3 regions · target `95%` in `15 min`.

2 CORE ENTITIES

5–9 min · records the failure probe needs

- **Artifact**(version, size, location, chunk list + hashes, checksum) — immutable
- **Rollout**(id, target group, desired version, **generation**, status running|paused|rolling back|done, cohort %, canary %)
- **Worker**(id, region, attempt/generation, actual version, state, heartbeat, bytes_done, error)
- state: idle → downloading → verified → draining → loading → ready(version, generation) → serving
- Readiness identifies the exact version, never just a healthy machine. Every report carries the generation it was made under.

3 API

9–13 min · one endpoint per requirement

- `POST /releases` {version, checksum, manifest}
- `POST /rollouts` {version, targets, cohort_pct, canary_pct} · `/rollouts/{id}/pause | resume | rollback` (rollback = new generation, desired = V1)
- `POST /workers/{id}/report` {generation, state, version, bytes_done, error} — older generation → ignored
- `GET /rollouts/{id}` → downloaded · verified · ready · serving, per region. The engineer acts on **serving**, never on downloaded.

4 DATA FLOW

13–20 min · one worker, end to end

- publish manifest + hashes → controller assigns generation G, picks a cohort inside the headroom →
- agent downloads by chunk, hash per chunk, resumable → whole-file checksum → controller: drain (in-flight finish, admission stops) →
- load into GPU memory + warm-up prompts → agent reports ready(V2, G) → router flips this worker → next cohort.
- **Before the first request hits V2:** verified, loaded, warmed, ready under the current generation, and enough V1 still serving.
- **Restart mid-download:** resume by verified chunk; a stale "done" with an old generation is discarded; duplicates change nothing.

5 HIGH-LEVEL DESIGN

20–33 min · the board, then the twist

- Rows: engineer → controller → origin → records · one seed per region · regional worker swarms · agent state machine, router (exact-version ready only), canary monitor.
- **Bandwidth first:** `100 GB = 800 Gbit → 80 s per copy at 10 Gbit/s` . 1,000 copies via the origin = `80,000 s ≈ 22 h` ; 950 ≈ 21 h. Direct download cannot make 15 min.
- **Redesign:** origin sends 3 copies (`240 s`), one per region; in-region fan-out on links the origin does not share (cache tree or chunked swarm, every worker uploads while downloading). Add load + warm-up to the clock.
- Trace the bytes: origin → seed → regional egress (next shared limit) → racks → workers. Feasible **only if** regional egress and load time hold: say what you would measure.

6 DEEP DIVES

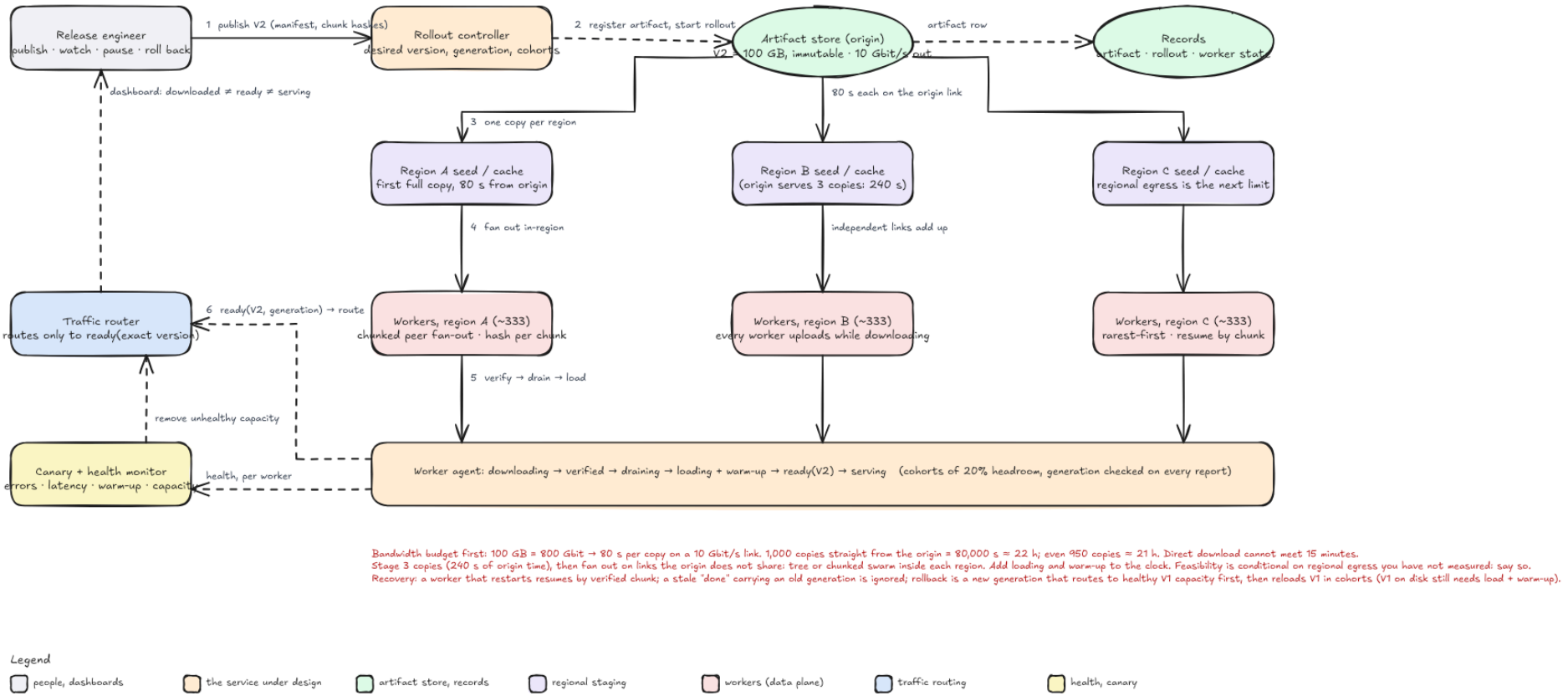
33–45 min · failure, rollback, recap

- **Load failures, ready-then-crash:** pull from routing; isolated vs systematic; stop signals: errors, latency, warm-up failures, capacity loss → pause.
- **Rollback:** healthy V1 capacity first, then reload V1 in cohorts; V1 on disk still needs load + warm-up; record the rollback generation so a late V2 "ready" cannot reverse it.
- **Dashboard:** 95% downloaded but 60% can serve → act on serving capacity.
- **Peers forbidden across regions:** seeding and in-region fan-out unchanged; only the inter-region hop is origin-fed.
- **Recap:** publish, distribute, verify, activate, route, recover. Guarantee: requests reach only verified, loaded, exact-version workers under the current generation. Bottleneck left: regional egress + load time. Validate: fan-out throughput and warm-up on one cohort.

Sources: interviewing.io mock with Kyle, 27 Sep 2026 (his rubric) · 1point3acres SD Q3 · hellointerview large-model distribution · rik-kisnah.ai/teach/gpu-ai/design-model-weight-distribution

The board at minute forty

back: the answer, as it should look on the shared screen



Interactive: link.excalidraw.com/readonly/3aIVH2GfBp7znDAs6Sya · source: 2026-09-27-model-rollout-answer-board.excalidraw

back