System Design on a Sovereign Stack - 2-page cheat sheet

Course companion - the four tests, carrying costs, observability, GitOps, honest costing.

The sovereignty question

- **The test:** 2 AM Friday: your cloud account is suspended - billing dispute, fraud flag, sanction, doesn't matter. What still works Monday morning? What can your team operate, restore, and move, and what's simply gone? If you can answer precisely per component, you're sovereign.

- **Thesis:** Cloud is not the enemy; irreversible dependence is. Sovereignty = controlling deployment, data location, and exit - without giving up the ability to ship.

- **If you're renting:** Know the rent, the landlord, and the eviction terms.

The four tests of sovereignty (1.1)

- **1. Open:** OSI-approved license; could fork tomorrow if the vendor pivots, is acquired, or triples the price. 'Source available' is not open.

- **2. Self-hostable:** Runs on hardware you choose, with no mandatory vendor control plane. Watch for enterprise-only clustering, backups, SSO - where many 'open-source' projects quietly fail.

- **3. Lock-in resistant:** Data and config move through portable interfaces, with a TESTED export/restore path. pg_dump exists; S3 API + mc mirror moves terabytes. No timed restore = no pass.

- **4. Data-sovereignty friendly:** You decide geography, which jurisdiction can compel access, and who holds the encryption keys. 'EU region label' with provider-held keys and default US replication does not pass.

Most stacks fail at least one test - the point is to make the failure visible and priced, not to shame you into self-hosting email.

Convenience has a carrying cost (1.3)

- **1. The meter:** Compute, storage, requests, API calls - the only line the sales deck shows.

- **2. Egress & transfer:** Data leaving the network, cross-AZ replication, NAT gateway per-GB fees. Often the largest surprise line.

- **3. Proprietary API surface:** Hours learning the vendor's IAM model, deployment quirks, query language.

- **4. Support:** Price the tier you'd actually buy - 'community support' is not a plan.

- **5. Production shaping:** Base price is single-zone, unreplicated, unbacked-up. Price what you'd actually run.

- **6. Operations labor:** Managed doesn't eliminate ops labor; it changes its shape. Price both sides.

- **7. Price-change & exit risk:** P(price hike) x cost of switching. A 30% hike on a service you can't leave in 6 months is a real expected cost.

Compare like with like (12.3)

- **Meters, not invoices:** An invoice is a bundle of choices someone else made. A meter is a unit: one vCPU-hour, one GB-month of NVMe, one GB of egress, one ops-hour.

- **Production-shaped:** If both sides aren't redundant, backed up, supported, and monitored, you're comparing a house to a photograph of a house.

Reference stack & escape hatches (1.5)

- **Approved list:** Ollama, PostgreSQL, Qdrant, MinIO, Redis/NATS, FastAPI + HTMX, Prometheus + Loki + Grafana, Caddy.

- **Escape hatches:** Every component choice ships with its replacement plan. Build vs rent is a portfolio decision, revisited per component.

Observability (Section 10)

- **RED (services):** Rate, Errors, Duration - for request-driven services. 'Are users succeeding, how fast, how often not.' If you can monitor three things, monitor these.

- **USE (resources):** Utilization, Saturation, Errors - for resources: CPUs, disks, memory, NICs, connection pools, queue depths.

- **SLOs:** Convert objectives into error budgets. Burn-rate alerts page on user impact, not on CPU graphs.

- **Loki:** Label discipline in pipeline config; avoid cardinality explosions; 30-day retention default with per-stream overrides.

- **Config as code:** Scrape jobs, recording rules (RED computed continuously), alerts - every alert carries severity, owner, runbook_url. Dashboards provisioned as JSON with deploy annotations - never click-opsed.

Deployment & GitOps (Section 11)

- **Source & CI:** Gitea = source-of-truth forge. Woodpecker CI pipelines.

- **Provision:** OpenTofu for provisioning; bootstrap k3s without snowflakes (every node reproducible).

- **Flux:** Pull-based reconciliation: Git is the ONLY writer of desired state; the cluster pulls; drift is detected and surfaced, never silently absorbed.

- **Never push from CI:** CI pushing with a cluster-admin kubeconfig makes your CI system a superuser target. Rejected - pull, don't push.

- **Release strategies:** Blue-green, canary, and rollback - rehearsed, not theoretical.

Data layer (Sections 6-8)

- **PostgreSQL:** Streaming replication + PITR backups. Foreign keys as guardrails. pg_dump = your lock-in-resistance proof.

- **Caching:** Cache-aside vs write-through; TTLs are policy, not decoration. Valkey.

- **Messaging:** Queues vs topics vs streams - choose by delivery semantics, not fashion.

- **Object storage:** S3-compatible (MinIO); media pipelines stream - never buffer large objects in the database.

Networking (Section 4)

- **Load balancing:** Algorithms, health checks, and sticky-session trade-offs - chosen per workload, not per default.

- **Service discovery:** Services find each other without hardcoded addresses.

- **Planes & gravity:** Control plane vs data plane: who decides vs who forwards. Hidden gravity: egress and transfer costs shape architecture.

Security (Section 9)

- **AuthN/AuthZ:** Authenticate every service, authorize every action, least privilege throughout.

- **Secrets:** Secrets in a vault with rotation - never baked into images or dumped in env.

- **Threat modeling:** Threat-model each component: trust boundaries, data classes, and what 'compromised' costs.

Capacity & cost (Section 12)

- **Workload model:** Model the workload first: request rates, data growth, peak multipliers.

- **Capacity:** Size with headroom - for peaks, failures, and deploys - then price the meters.

- **Exit strategy:** One-page exit plan per critical component: export path, restore drill date, replacement candidate.

Runbook habit (10.6)

- **Alert contract:** Every alert: severity, owner, runbook_url. A page without a runbook is a guess with a pager.

- **Drill restores:** Timed restore drills. Lock-in resistance is demonstrated, not declared.

Capstone habit (Section 13)

- **End-to-end ownership:** Workload model -> capacity with headroom -> honest meter-based cost comparison -> documented exit plan -> architecture review. Prove you can own the whole stack, on paper and in practice.