Architecture review checklist

From ArchitectureCompass (https://architecturecompass.com). Free to use and adapt.

Use this before a design is approved, and again before go-live. Each item should have an answer someone can point to, not just a tick.

1. Purpose and context

- [ ] The business goal and the problem being solved are written down in one paragraph.
- [ ] Users, systems and organizations that interact with the system are identified (a context diagram exists).
- [ ] Constraints are listed: budget, deadlines, regulations, mandated technology, team skills.
- [ ] Out-of-scope items are stated explicitly.

2. Quality attributes (non-functional requirements)

- [ ] Expected load is quantified: users, requests per second, data volume, growth over three years.
- [ ] Latency targets are stated as percentiles (for example, p95 under 300 ms).
- [ ] Availability target is agreed, with what counts as "down".
- [ ] Recovery point objective (RPO) and recovery time objective (RTO) are agreed for each data store.
- [ ] Security, privacy and compliance requirements are listed (data classification, residency, retention).
- [ ] Each quality attribute has a way to test it before launch.

3. Structure and boundaries

- [ ] The chosen architecture style is named, with the alternatives considered and why they were rejected.
- [ ] Module or service boundaries follow business capabilities, not technical layers.
- [ ] Each data store has exactly one owning component; others use its interface.
- [ ] Dependencies point in an agreed direction; there are no cycles between components.
- [ ] Shared libraries and shared databases are justified, not accidental.

4. Data

- [ ] The main entities and who owns them are documented.
- [ ] Consistency needs are clear: what must be transactional, what can be eventually consistent.
- [ ] Schema changes can be deployed without downtime.
- [ ] Personal data is identified, minimized, and can be deleted on request.
- [ ] Backups exist **and a restore has been tested**, with the time it took recorded.

5. Integration

- [ ] Every interface has an owner, a versioning approach and a contract (OpenAPI, AsyncAPI, Protobuf or similar).
- [ ] Synchronous calls have timeouts, limited retries with backoff and jitter, and a circuit breaker where needed.
- [ ] Operations that create or move something of value are idempotent (for example, with idempotency keys).
- [ ] Messages and events are designed for at-least-once delivery: consumers handle duplicates.
- [ ] Failed messages go to a dead-letter queue that someone monitors.

6. Security

- [ ] Authentication and authorization are designed for users and for service-to-service calls.
- [ ] Data is encrypted in transit and at rest; key ownership is clear.
- [ ] Secrets live in a secrets manager, never in code or images.
- [ ] A threat model has been done for the main data flows.
- [ ] Least privilege is applied to people, services and pipelines.
- [ ] Destructive operations (bulk deletes, data wipes) have safeguards: confirmation, soft delete, rate limits.

7. Reliability and resilience

- [ ] Single points of failure are identified, and each is removed or accepted in writing.
- [ ] The blast radius of a failure is limited (cells, shards, bulkheads, tenant isolation).
- [ ] The system degrades gracefully when a dependency is down (fallbacks, cached data, queues).
- [ ] Capacity has headroom for known peaks, and autoscaling has been tested under load.
- [ ] Failure scenarios have been tested on purpose (instance loss, dependency outage, region loss if required).

8. Operations and observability

- [ ] Logs, metrics and traces are in place, with correlation IDs across services.
- [ ] Alerts are based on user-facing symptoms (error rate, latency), with a named on-call owner.
- [ ] Monitoring and incident tooling do not depend on the system they monitor.
- [ ] Runbooks exist for the most likely failures.
- [ ] Dashboards show the service's health at a glance.

9. Delivery and change

- [ ] Builds, tests and deployments are automated and repeatable.

- [ ] Code **and configuration** roll out in stages (canary or rings), with automatic or one-step rollback.
- [ ] Feature flags or kill switches exist for risky features.
- [ ] The same artifact is promoted through environments; no manual changes in production.

10. Cost

- [ ] A cost estimate exists for expected and peak load.
- [ ] The biggest cost drivers are known (compute, storage, data transfer, per-request charges, licences).
- [ ] Resources are tagged so costs can be attributed to an owner.
- [ ] Budgets and alerts are set.

11. Decisions and documentation

- [ ] Significant decisions are recorded as architecture decision records (ADRs).
- [ ] Diagrams exist at the right levels (context, containers, key components) and are current.
- [ ] Open risks and assumptions are listed, each with an owner.
- [ ] The conditions that would trigger revisiting the architecture are written down.