DESIGNGURUS.IO

The Microservices Interview Cheat Sheet

A six-step framework for answering any microservices system design question, ten worked case studies compressed to one card each, and the twelve answers that mark a candidate as junior.

- The six steps, with minute budgets for a 45-minute round
- Boundaries, communication, consistency, failure, operations
- Ten case studies: the calibration, the spine, the hard part
- Twelve red flags, each with its fix

By Arslan Ahmad

THE FRAMEWORK

Six steps, in dependency order

A memorized framework does two jobs. It keeps you from skipping anything load bearing, and it puts the decisions in dependency order so each step's output feeds the next one. The budgets assume a 45-minute round with about 35 minutes of design time.

1

Requirements and scale

5 min

Functional scope as a list of verbs. Then the numbers you will actually use: requests per second, data sizes, latency. Ask for team and org context, because that decides how much splitting is justified.

Close with the two or three requirements that drive the design.

2

Boundaries and data ownership

5 to 7 min

Draw services by capability, not by noun. Say the justification out loud as you draw each box: what it owns, why it is separate, which single service writes each datum.

Fewer boxes with defended lines beat many boxes with none.

3

Communication per edge

5 min

One question per arrow: does the caller need the answer to proceed? Then protocol, events fat enough to avoid callbacks, a gateway at the front, and the coordination style for multi-step flows.

Every service off the request path is latency and availability reclaimed.

4

Data and consistency

7 to 10 min

The depth step. Find every place two services' data must agree, then apply the machinery by name: sagas, the outbox, idempotency, read models.

One honest sentence about what is eventually consistent and who sees it.

5

Failure handling

5 to 7 min

Pick the two scariest edges and run them end to end. Timeout number, retry discipline, breaker and bulkhead, and what the user sees.

One full failure walkthrough beats five pattern names.

6

Operations

3 to 5 min

One paragraph: canary deploys with automated metric comparison, RED dashboards, symptom-based alerts, one trace ID end to end, mTLS and least privilege.

Even one sentence here banks most of its value.

STEPS 2 AND 3

Boundaries, then one decision per arrow

Drawing the boxes

- **By capability, not by noun.** A capability is something the business does, like taking payments. A noun is just a thing, like "User." Entity-named services produce a five-hop tour for every operation.
- **One writer per datum.** Exactly one service is allowed to write each piece of data. Say it as you draw each box.
- **Justify each line out loud.** What it owns, why it is separate. Team ownership, divergent profiles, and change patterns are the three real reasons.
- **Fewer boxes with defended lines beat many boxes with none.** If the system should not be microservices at the stated scale, say so and propose the modular monolith. That sentence is a scoring event.

Deciding each edge

- **The one question:** does the caller need the answer to proceed? That decides sync or async. Nothing else does.
- **Protocol.** REST for outside clients, gRPC between your own services.
- **Events published where the facts happen,** fat enough that consumers never call back for more.
- **A gateway at the front.** One entry point, authentication at the edge, no business logic in it.
- **A coordination style per multi-step flow.** Orchestration means one service directs. Choreography means services react to each other's events.

The sentence that scores

- "Every service off the request path is latency and availability reclaimed. And next quarter's fraud consumer subscribes without Checkout changing."

STEPS 4 AND 5

The depth step, then the failure walk

The consistency toolkit

- **Saga.** A sequence of local transactions across services, each with an undo step. Name every compensation. Put the riskiest step last. It gives you atomicity, consistency, and durability, but not isolation.
- **Outbox.** On every seam where you write to your database and then publish an event. The event saves in the same transaction as the data change, and a relay publishes it afterward. Without it you have a dual write, and dual writes lose data.
- **Idempotency.** A key minted once per logical action, stored atomically with the effect. This is what makes a retry safe on a payment.
- **Read models.** A read-only copy shaped for one query, fed by events, for anything that needs data from more than one service.
- **An honest sentence** about what is eventually consistent and who sees it. Add read-your-writes wherever the user would notice.

The walk every scary edge deserves

1. You draw an arrow between two services.
2. Ask it before anyone else does: what happens when this call is slow or fails?
3. A timeout, with a stated number, taken from the dependency's p99 latency.
4. A retry only if repeating the work is safe, with jitter so retries do not all land at once.
5. A breaker that opens after repeated failures, so a dead dependency gets room to recover.
6. A bulkhead capping how much of your capacity that one dependency can consume.
7. A fallback or reduced response, and then what the user actually sees on screen.

The habit that reads as senior: run this walk unprompted, every time you draw an arrow. One full walkthrough beats five pattern names.

STEP 6 AND DELIVERY

Operations, driving, and the follow-ups

Operations in one paragraph

- **Canary deploys** send the new version a small slice of traffic first, with automated metric comparison before a full rollout.
- **RED dashboards** track rate, errors, and duration for every service.
- **Symptom-based SLO alerts** fire on what users feel, not on raw CPU.
- **One trace ID** follows a request from the gateway through every event it triggers.
- **mTLS and least privilege:** every service proves who it is, and reaches only what it needs.

Three meta-skills that sit on top

- **Announce the plan** in the first minute. It costs ten seconds and buys the interviewer's agreement on where to go deep.
- **Follow the interviewer's energy.** The framework is your map, not your script. Fighting for the agenda loses the points structure was supposed to win.
- **Timebox visibly.** "I'll fix these two services and revisit if needed." An unfinished consistency step costs more than an imperfect boundary step.

The four follow-ups that decide the grade

- **"Two customers race for the last unit."** Strong consistency at the contention point, eventual everywhere else. First commit wins, the second gets an honest failure.
- **"Where exactly can this be lost?"** Walk the seams one at a time and name a mechanism at each. Then state the conclusion: there is no silent-loss seam.
- **"Why not fewer services? More?"** Team ownership, divergent profiles, change patterns. Name your first consolidation candidate.
- **"What breaks first at 10x?"** Have the answer before they ask, with a named treatment for each.

CASE STUDIES

Ten questions, one card each

1 E-commerce Order System

Cart through payment, fulfillment, and notifications. About two million orders a month, 300 engineers.

THE HARD PART Transactional integrity and fan-out

Calibration	About 8 orders/sec at peak, so moderate scale. Say that the hard parts are correctness and fan-out.
Boundaries	Five capability services. No User service. One writer per datum.
Communication	Three sync hops on the critical path. One fat event feeds everything downstream.
Deep dive	Orchestrated saga, outbox everywhere, keyed charge, read-model order history.
Failure runs	Payments down (breaker, then a product choice). Broker down (the outbox holds, consumers catch up).

2 Food Delivery Platform

Customers order from restaurants, drivers deliver, everyone watches the order move live. A million orders a day across 50 cities.

THE HARD PART Three actors, live state, physical-world contention

Calibration	Orders are about 12/sec. Driver pings are about 12,500/sec, so location dominates and gets its own service.
Spine	An order state machine publishing an event per transition. Every view is a consumer.
Real time	WebSockets or SSE out to three actor apps. Pings come in through queues, and losing a ping is free.
Contention	Offers fan out. Acceptance is one local transaction on one row. Same for the restaurant accept.
Human failures	Timeouts on state transitions are business logic. Escalate, re-dispatch, compensate approximately.

Ten questions, one card each

3 Ride-Hailing Platform

Riders request, nearby drivers are offered the trip, both watch it live, the rider is charged at the end. 20 million rides a day.

THE HARD PART Match quality and metered money

Calibration	About 700 rides/sec at peak, sharded by city. Throughput is easy. Match quality and money are hard.
Matching	Spatial index for candidates, then a batched assignment solve over a 1 to 2 second window. Greedy is the fallback.
Pricing	Surge per zone from a rolling window. Quoted with an expiry, pinned on the trip, never re-read at charge time.
Metered fare	The GPS trace is financial data. The fare finalizes only after the trace is complete, capped at 60 seconds.
Money failures	A decline becomes a receivable, not a rollback. The driver is paid from a separate ledger either way.

4 Ticket Booking Platform

Named and numbered seats. When a popular show goes on sale, 50,000 people hit it in the same second for 20,000 seats.

THE HARD PART A hard never-oversell invariant under a thundering herd

Calibration	50,000 reads/sec at the on-sale instant, a few hundred writes/sec at most. Two different problems.
Boundaries	Seat Inventory (one partition per event) split from Seat Map Reads (cached, stale, fanned out).
The herd	A virtual waiting room at about 1,000 admissions/sec, honest queue position, per-account and per-card caps against bots.
The hold	Available to held to sold. TTL enforced by the server, expiry applied lazily on the next claim.
Never oversell	One conditional write on one seat row. One row updated wins, zero rows updated loses.

Ten questions, one card each

5 Payments and Wallet Platform

Users top up from a card, hold a balance, pay merchants, withdraw to a bank. Every number must be explainable later.

THE HARD PART Auditability, and a third party you cannot roll back

Calibration	About 580 movements/sec is easy. Seven billion immutable entries a year and 12,000 balance reads/sec are the real constraints.
Ledger	Double-entry, append-only, one writer. Corrections are reversing entries, never edits.
Balance	Derived. A projection written with the entries and provably recomputable.
Once in effect	At-least-once delivery plus an idempotency key with a unique database constraint. On a provider timeout, query status by key instead of resending.
Reconciliation	A daily three-way match against the settlement file and the bank, with break age as a paged metric.

6 Chat and Presence

One-to-one and group conversations, delivery and read receipts, and a reliable indicator of who is online. 10 million concurrent connections.

THE HARD PART Stateful nodes, and ordering as correctness

Calibration	10M connections at about 100k per node is about 100 gateway nodes. Presence fan-out, not chat, is the biggest number.
Boundaries	One thin stateful gateway holding sockets. Chat, Fan-out, Presence, and Receipts keep their state in stores you can restart around.
Ordering	Monotonic sequence numbers from the conversation's single writer. Client clocks are never used to sort.
Delivery	Store before push. At-least-once plus client dedupe on message id equals exactly-once for the user.
Presence	Heartbeats with a TTL that expires on its own. Slightly wrong and cheap, batched per node, with the fan-out damped hard.

Ten questions, one card each

7 Video Upload and Streaming

Creators upload source files, you transcode each into several qualities, viewers stream worldwide. 500 hours uploaded every minute.

THE HARD PART Work measured in minutes, not milliseconds

Calibration	50 API requests/sec is nothing. 270 Gbps of ingest and about 150,000 busy cores are the system.
Write path	One presigned URL per part, resumable straight to storage. Your API issues credentials and reacts.
Pipeline	A step graph (probe, segment, transcode, package, thumbnail, publish) with per-step retry and per-step idempotency.
Handoff	Messages carry object keys, never payloads. Storage is the bus between stages.
Partial success	The minimum ladder publishes, missing renditions are absent from the manifest, top rungs are encoded lazily.

8 Notification System

Any service can request that a user be notified. Delivery over push, email, and SMS, with user preferences. Around 100 million a day.

THE HARD PART Infrastructure, not a product: fan-out and dedupe

Shape	One bounded context, staged: ingest, route by preferences, render, then per-channel senders.
Spine	Queues between every stage: durability, decoupling, load leveling, per-channel scaling.
Priority	Two tiers per channel. Transactional drains first, so campaigns never delay password resets.
Dedupe	Layered: an ingestion idempotency key, per-stage dedupe, a send-log commit. Admit the provider-call window.
Providers	A breaker per provider, multi-home failover, a DLQ for poison messages, and a kill switch by design.

Ten questions, one card each

9 Migrating a Legacy Monolith

A twelve-year-old monolith, 200 engineers, twice-weekly terrifying deploys, and a CTO who has already announced microservices.

THE HARD PART Organizational risk, and a trap in the question

Opening move	Evidence before boxes: deploy contention, divergent profiles. Success is deploy independence, not service count.
Strategy	Strangler fig via a routing layer. Reject the parallel rewrite, with reasons.
Sequencing	Notifications and Reviews first to learn cheaply, Search second, Payments and Booking last. The first extraction buys the playbook.
Data rule	Separate tables in place, a CDC transition, shadow reads. Shared database reads mean the extraction is not done.
End state	Allowed to be services around a healthy core. A migration that cannot stop is a rewrite in disguise.

10 Critique This Architecture

Twelve services and 30 engineers, one shared database, a six-hop synchronous order chain, five payment retries, business logic in the gateway.

THE HARD PART Ranking findings, and delivering them well

The method	Four passes: follow the data, follow a request, follow a failure, follow the responsibilities.
Critical findings	Retries without idempotency (double charge). Database in liveness probes (one blip kills all twelve). Shared database (a distributed monolith).
Structural findings	Entity boundaries force a six-hop chain. Business logic in the gateway. Twelve services for 30 engineers.
Direction	Consolidation, not more microservices discipline.
Delivery	Severity-ranked, each with a fix attached, plus concessions and questions.

RED FLAGS

Twelve answers that sound junior

Nearly every red flag is one of three shapes: a benefit stated without its cost, a mechanism named without its consequence, or a slogan where a judgment was requested.

1. **"Microservices, because they scale better."**

   FIX Microservices solve organizational problems: independent deployment for multiple teams. They cost you distribution: latency, consistency, operations.

2. **Monolith used as a synonym for legacy mess.**

   FIX Bring up the modular monolith yourself, then name the evidence that would justify splitting it.

3. **One service per noun: User, Order, Product, Payment.**

   FIX Split by capability, not entity. State who owns which data as you draw each box.

4. **Many service boxes, one shared database cylinder.**

   FIX One private store per service. Meet cross-service needs with events and owned copies, and say the coupling reason out loud.

5. **"We'd add retries" as the universal failure answer.**

   FIX Backoff, jitter, budget, retries at one layer only, idempotent operations only, and a breaker for sustained failures.

6. **A payment retry with no idempotency story.**

   FIX An idempotency key, minted once per logical action, stored atomically with the effect.

RED FLAGS

Twelve answers that sound junior

7 **"It returns an error and we show an error message."**

FIX Distinguish fast, slow, and ambiguous failures. Then run the loop: detect, respond, contain, degrade.

8 **Everything synchronous, or everything events.**

FIX Ask the one question per arrow. Does the caller need the answer to proceed?

9 **"We'd use Kafka" with no job attached.**

FIX Name what the tool does on that specific edge, and what it costs.

10 **Calling a saga compensation a "rollback."**

FIX Compensations are business actions. Put the riskiest step last, and name isolation as the missing letter.

11 **"Kubernetes handles it."**

FIX Consume the platform's real guarantees by name (scheduling, self-healing, rollout, discovery) and own everything above them.

12 **"We'd monitor it and roll back if something goes wrong."**

FIX A canary with automated metric comparison, symptom-based SLO alerts, trace IDs end to end, and a named detection-to-rollback path with times.

The pattern behind all twelve

- Every fix is the same move: **attach the cost, the consequence, or the evidence.** The sentence shape is "X, because Y, and it costs Z, so I'd watch for W." Rehearse until the cost arrives in the same breath as the benefit, unprompted.

PRACTICE

How to actually get better at this

Three drills, in order of value

- **Talk, do not read.** Set a 45-minute timer and deliver a whole answer out loud to an empty room. Reading a good answer feels like learning, but the interview asks you to produce one.
- **The self-audit drill.** Record yourself answering five rapid questions, then count how many answers included a cost or a failure mode without being asked. The target is five of five.
- **Collect your own follow-ups.** After each run, write the three questions you would have asked yourself, and answer them next time. That conversation is where the grade is decided, and almost nobody rehearses it.

The eight question families

- Work the families, not individual answers. Each has its own hard part: **transactional integrity** (e-commerce), **physical-world contention** (delivery, ride-hailing), **a hard invariant** (ticketing), **auditability** (payments), **stateful connections** (chat), **long-running compute** (video), **fan-out** (notifications), and **organizational risk** (migration).

Go deeper

Grokking Microservices Design Patterns teaches the mechanisms behind every pattern named here, one at a time, with worked examples.

Grokking the System Design Interview covers the full round. **Grokking System Design Fundamentals** is the place to start if the vocabulary here was new.

designgurus.io