

Shailendra Singh

SENIOR BACKEND ENGINEER

srajawat024@gmail.com | Bangalore, India | [linkedin.com/in/shailendrasinghw3](https://www.linkedin.com/in/shailendrasinghw3) | github.com/Shailu-s

PROFILE

Senior backend engineer, 6+ years, building scalable distributed systems and microservice architectures in Go, TypeScript, Python, and Node.js, with hands-on DevOps: Docker, Kubernetes, Terraform, and the CI/CD that ships them. Comfortable across the full stack of a backend: API and database design, internal auth and middleware layers, concurrency and caching, async job processing, idempotency, the infrastructure it deploys to, and the observability needed to run it in production.

EXPERIENCE

Qiro Finance · Senior Backend Engineer

04/2026 – Present · Bengaluru, India

- Designed and built the production accounting system behind a revolving line of credit product, RLOC: a borrower draws money against a limit, repays it, and can draw again, like a credit card rather than a single fixed loan. Modelled the money lifecycle (borrowing, repayment, missed payment, recovery) as an explicit state machine over an append-only ledger.
- Made RLOC's money math provably correct: arbitrary-precision integer arithmetic rather than floats, deterministic replay from the event log, atomic Postgres transactions, and invariants enforced at the database layer rather than in application code, so two workers racing on the same event cannot both commit.
- Built RLOC's event ingestion pipeline from two independent sources, push (webhooks) plus a polling scanner as a safety net, with idempotent writes, automatic gap detection and backfill, and rollback handling for upstream events later invalidated, so no event is lost, duplicated, or applied twice.
- Cut latency on the customer-facing portfolio and balance APIs by caching an expensive multi-source aggregation, with per-field TTLs set from how fast each value could legitimately change instead of one blanket expiry.
- Built the CI/CD pipeline in GitHub Actions for the backend monorepo: pull requests run typecheck and the integration suite against an ephemeral Postgres service container in Docker, and merges build, validate required environment configuration, and deploy the API and web apps to AWS automatically, with path filters and concurrency gating so superseded runs cancel instead of racing.

Tokamak Network · Senior Backend Engineer

07/2024 – 03/2026 · Remote

- Worked on Tokamak Rollup Hub, a Rollup-as-a-Service product letting developers deploy their own custom L2 chains with plug-and-play bridges and block explorers, instead of standing up that infrastructure by hand. Contributed to the Go SDK and CLI that drives it: stands up cloud infrastructure, deploys the service stack to Kubernetes with Helm, and tears it down cleanly, replacing what used to be days of manual setup.
- Built the candidate registration flow: a multi-step deployment sequence that registers a new stack with the network, and can resume from a standalone command if it breaks partway through, instead of leaving the operator with a half-finished deployment and no way forward. Added verification checks so a bad deployment can't silently pass as valid, plus incremental timeout handling with backoff so a slow step retries instead of failing the whole flow outright.
- Built the TypeScript backend behind the platform: authentication, the Postgres schema and access layer, and all the calls out to external services, with retries and idempotent handlers so a flaky upstream failure couldn't corrupt deployment state or get applied twice.
- Worked on the backup manager for deployed chains (automated state and database backups with a restore path) and the platform dashboard's monitoring integration, including making alerting configuration optional rather than mandatory so setup isn't blocked on a channel an operator doesn't use.

Syvora · Software Engineer

12/2020 – 07/2024 · Remote

- Built the core auction path of a Go microservices system for a client, Blockswap Labs, optimising real-time bidding for hundreds of concurrent participants: a deadline-bounded auction that fans out concurrent bid requests to multiple upstreams, collects responses under a hard deadline via context cancellation, and returns the best bid received so far rather than missing the deadline waiting for a slow upstream.
- This was a genuinely latency-sensitive system: the whole auction had to close inside a fixed 12-second window with no room for a wrong answer, since missing it meant the round was wasted outright. I tuned the cutoff by analyzing real latency distributions, moving it from 10 seconds into the window to 8, which traded about 5% of late-arriving bids for a hit rate that went from 94% up to 99.5%.
- Designed that system as independent modules communicating over gRPC, so new providers could be added without modifying core auction logic: an explicit trade-off accepting inter-process overhead for isolation and pluggability. Made it tolerant of partial failure, dropping an upstream that times out or returns malformed data from that round instead of failing the auction, and debugged real goroutine races in the fan-out/fan-in path.
- Built serverless analytics pipelines on AWS Lambda for that same client: independent functions tracking participant status, throughput, and performance metrics, triggered off upstream events and feeding a SQL database behind an internal reporting dashboard.
- Built the real-time order matching engine in Go for a derivatives exchange, designed and benchmarked for 50K orders/sec on an in-memory price-level order book, holding sub-millisecond match latency by keeping disk off the hot path: fills and settlement are written asynchronously downstream, with crash recovery that rebuilds the book on restart from persisted history.
- Built the asynchronous event-processing pipelines on RabbitMQ ingesting high-throughput transaction and event data into PostgreSQL and MongoDB, and implemented high-concurrency TypeScript workers for order management, settlement, and transaction orchestration.
- Implemented peer-to-peer state synchronisation with leader election, keeping matching consistent across nodes without a central coordinator.

PROJECTS

Order Execution Service (NestJS, TypeScript). A conditional order engine: stop-loss, take-profit, and stop-limit orders sit dormant until a live price feed crosses their trigger, then convert to a market or limit order and execute. Prices broadcast over WebSocket rather than polling, so trigger checks run against the same stream clients see. Order state, execution, and the price feed are separate modules so trigger logic can be tested without a live feed. 99 tests across 7 suites, unit and integration.

shortn (Go). A URL shortener built to find out where a read-heavy service actually breaks: load-tested with k6 against a real Postgres and Redis stack, redirects measured at 37,893/sec peak at 2.32ms p95 with zero errors, roughly 570 million redirects/day at realistic traffic patterns. Writes measured at 12,141/sec; a nginx-fronted read/write split documents its own bottlenecks (Redis memory, single-instance) once traffic pushes past that.

SKILLS

Languages. Go, JavaScript/TypeScript, Python, Rust, SQL

Frameworks. Node.js, Fastify, Django, FastAPI

Databases. PostgreSQL, MongoDB, Redis, MySQL, DynamoDB, SQLite, Elasticsearch

Infrastructure & DevOps. Docker, Kubernetes, Helm, Terraform, AWS, GCP, GitHub Actions CI/CD, Prometheus, Grafana

EDUCATION

Bachelor of Engineering, Computer Engineering, Institute of Engineering and Technology, DAVV

07/2015 – 03/2019 · Indore, India