System Architecture

1) Architecture Summary

DRIVE-U is a cloud-native, microservices-oriented platform that securely connects car owners and RTO-verified drivers via mobile apps, providing real-time GPS tracking, automated matching, transparent pricing, and operational analytics — designed for high availability, rapid scaling, and regulatory compliance.

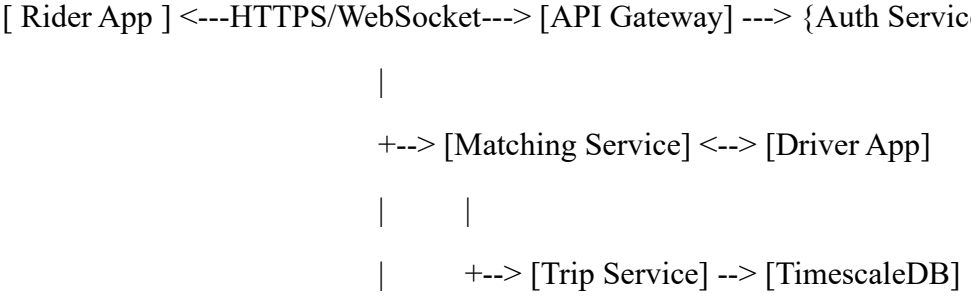
2) High-level logical architecture (layers)

1. **Client Layer**
    - Rider Mobile App (iOS / Android — Flutter or React Native)
    - Driver Mobile App (native-capable; background GPS)
    - Web Admin / Ops Dashboard (Next.js / React)
2. **Edge & API Layer**
    - API Gateway (Auth, Rate limiting, TLS termination)
    - WebSocket / Realtime Gateway (for live trip updates)
    - CDN for static assets
3. **Microservices Layer**
    - Auth & User Service (JWT, sessions, RBAC)
    - Driver Verification Service (RTO integration, KYC/OCR pipeline)
    - Matching & Dispatch Service (real-time allocation engine)
    - Pricing Service (dynamic pricing / surge)
    - Trip Management Service (state machine for trip lifecycle)
    - Notifications Service (push / SMS / email)
    - Payment & Billing Service (gateway integration, settlements)
    - Ratings & Feedback Service
    - Analytics & Reporting Service (event processing)
4. **Data Layer**

- OLTP: PostgreSQL (users, drivers, trips, transactions) + PostGIS for geospatial queries.
- Time-series / Telemetry: TimescaleDB or InfluxDB (high-resolution GPS/telemetry).
- Cache / Session: Redis (caching, locks, leader election).
- Blob Storage: S3 (documents, driver photos, logs).
- Search & Analytics: Elasticsearch (fast search, logs), Data-warehouse (Redshift / BigQuery) for BI.
- Event Bus: Kafka (telemetry, audit trail, async pipelines).

5. **Integration / External**
   - RTO Verification API (secure server-to-server integration).
   - Maps & Routing (Google Maps / Mapbox for directions, distance, and SDKs).
   - OCR / Identity: Google Vision / AWS Textract or custom OCR.
   - Payment Gateway: Razorpay / Stripe / Paytm.
6. **Platform & Infra**
   - Kubernetes (EKS/GKE/AKS) for orchestration; containers for each microservice.
   - Terraform (IaC), GitHub Actions (CI/CD).
   - Observability: Prometheus + Grafana (metrics), ELK (logs), Jaeger / OpenTelemetry (tracing).
   - Secrets: Vault / Secrets Manager.
7. **Security & Compliance**
   - End-to-end TLS, encrypted at rest, RBAC, audit logging, data residency controls.

3) Component diagram (textual / flow)



|

+--> [Pricing Service] --> [Postgres]

|

+--> [Driver Verification Service] --> [RTO API / OCR]

|

+--> [Notifications Service] --> (Push/SMS)

|

+--> [Payments Service] --> (Payment Gateway)

|

+--> [Analytics Service] --> Kafka --> Data Warehouse

(Web Admin connects to API Gateway -> Admin services; Operator tools read from BI / Elasticsearch.)

4) Booking/Trip sequence (numbered steps)

1. Rider requests booking (pickup/drop, time, preferences) from Rider App.
2. Request reaches API Gateway → Auth verification → Trip Service creates tentative trip (POSTgres).
3. Matching Service queries nearby verified drivers (PostGIS/Redis + recent telemetry) and ranks by ETA, rating, acceptance probability; Pricing Service computes fare.
4. Selected driver receives request via Driver App (WebSocket / Push). Driver accepts → Trip Service confirms and changes state to "Assigned".
5. Driver location is streamed (1–5s) to Realtime Gateway → TimescaleDB (telemetry) + Trip Service updates route. Rider sees live tracking + geo-fence alerts.
6. On trip completion, Trip Service finalizes fare, Payment Service processes payment, Ratings service collects feedback, Analytics service ingests event stream.
7. Audit logs / receipts stored to S3; periodic reconciliation and settlements executed by Billing subsystem.

5) Real-time & scale considerations

- **Telemetry ingestion:** Use Kafka or MQTT for high-throughput GPS event ingest. Buffer on the gateway, batch writes to TimescaleDB.

- **Matching latency:** Target sub-300ms response for matching queries; use spatial indexes (PostGIS) + Redis caching for hot regions.
- **Auto-scaling:** Horizontal pod autoscaling for matching, trip, and websocket gateways based on CPU/requests/consumer lag.
- **Throughput planning (example):** design for N concurrent trips; matching service must handle peak QPS — estimate from expected city users (adjust after MVP analytics).

6) Security, verification & fraud prevention

- **Driver KYC:** RTO API + OCR + manual human review workflow for edge cases. Store minimal PII; encrypt docs in S3.
- **Session security:** Short-lived JWTs, refresh tokens stored securely, device fingerprinting for driver app.
- **Trip safety features:** geo-fencing, route deviation alerts, emergency SOS routed to operations with trip snapshot.
- **Fraud detection:** Real-time analytics to detect suspicious patterns (multiple accounts, GPS spoofing, unusual cancellations).
- **Audit & compliance:** Immutable event logs in Kafka + cold storage for audits.

7) Operational & observability requirements

- **Metrics:** request latency, matching success rate, acceptance rate, GPS latency, failed verifications.
- **Alerts:** driver KYC failures, high trip cancellation, critical errors in dispatch.
- **SLA Goals (example):** 99.9% availability for core API; 99.95% for critical services with multi-AZ deployments.
- **Business KPIs:** avg ETA, conversion (booking→start), driver utilization, ARPU, churn.

8) Deployment topology & DR

- **Multi-AZ clusters** with node pools for critical (matching, auth) and batch (analytics) workloads.
- **Backups:** RDS automated backups + WAL archiving; S3 lifecycle policies.

- **DR:** warm standby in secondary region; cross-region replication for essential state (read replicas, Kafka mirroring).

9) Non-functional requirements (NFRs)

- **Availability:** ≥99.9% for core features.
- **Latency:** end-to-end booking <1s (API), matching <300ms, real-time location updates <5s.
- **Scalability:** support thousands of concurrent trips per city; scale linearly across cities.
- **Security & Privacy:** encrypted PII, role-based access, compliance with Indian data rules.
- **Maintainability:** small, well-scoped microservices, clear API contracts, automated tests.

10) Roadmap: technical milestones (prioritized)

1. **MVP:** Monolith or small set of services — Auth, Trips, Matching (rule-based), Postgres, Redis, basic WebSocket, Google Maps. (0–3 months)
2. **Verification & Payments:** RTO integration + OCR + Payment gateway. (3–6 months)
3. **Scale & Observability:** Break out microservices, add Kafka/TimescaleDB, Prometheus/Grafana, autoscaling. (6–9 months)
4. **Intelligence & Optimization:** ML-based matching, predictive ETAs, dynamic surge tuning, BI platform. (9–15 months)

11) Key tradeoffs & assumptions

- **Assumptions**
    - RTO verification is available via an API or partner program; if not, add stronger manual KYC steps.
    - Initial user scale is moderate (500–5,000 users per pilot city); architecture tuned for gradual scaling.
    - Google Maps or equivalent is acceptable for routing (costs will scale with usage).
- **Tradeoffs**
    - Managed services (AWS RDS, S3, Managed Kafka) speed development but increase operational costs.

- Immediate microservices decomposition increases complexity; start with a modular monolith for the MVP if team is small.