University of Colorado Boulder

Data Center Scale Computing (DCSC) | Fall 2025

Safer Ride — Risk-Aware Bicycle Routing

Project Proposal

Team: Simran Jadhav & Rohan Jain

Date: December 3, 2025

Motivation. All of this began with a close friend's bike accident. He ultimately required stitches for an injury to his chin. Helping riders make better route decisions is the main goal. These pathways reduce the amount of time spent on dangerous streets. Overall, we try to minimize any additional travel time. It all uses cloud-based tools for mapping locations and open city data.

Title, Participants, and Project Goals

Title: Safer Ride — Risk-Aware Bicycle Routing
Participants: Simran Jadhav & Rohan Jain
Roles (initial plan): Simran will lead data/ETL and frontend integration; Rohan will lead backend routing and ops. Roles may shift as milestones evolve.

Project Goals (clear to the class)

We will build a web application that recommends *safer bicycle routes* by combining crash history, 311 hazards, and bicycle-infrastructure layers.

Why compelling. This project turns open civic data into safer everyday choices for riders we personally know, making it both socially useful and technically engaging beyond a typical lab.

Planned deliverables

- The app will return two options for any origin–destination pair: **Fastest** (time-optimal) and **Safer** $\min\{\, time + \beta \cdot risk \,\}$.
- The UI will include **place search with dropdown suggestions** (geocoding) and be accessible with keyboard navigation and ARIA labels.
- A **risk overlay** (legend + animated start/end markers) will be rendered on Mapbox GL JS.
- We will constrain scope to the **Denver metro** to keep ETL feasible and data quality high.

Success criteria (how we will judge completion)

- Median API latency will be $<$ **800 ms** for cached requests; cold start will be $<$ **2.5 s**.
- The safer route will reduce exposure to high-risk segments by $\geq$**25%** with $\leq$ **10%** extra travel time vs. the fastest route.
- Lighthouse accessibility score will be $\geq$**95**, and custom layers will persist across theme/style changes.

Specific List of Software and Hardware Components

Frontend

- We will use static **HTML/CSS/JavaScript** (no React) with **Mapbox GL JS** for basemap rendering, autocomplete, and theme control (dark/light).

Backend

- We will implement **FastAPI** (Python) as a containerized function on **AWS Lambda**, exposed via **Amazon API Gateway**.
- Endpoints will include: `POST /route` (fastest+safer polylines and metrics), `GET /risks` (overlay/tiles), and `GET /health`.
- Routing will use A*/Dijkstra on an OSM-derived bike graph; the safer objective will be $time + \beta \cdot risk$ with tunable β.

Data & ETL

- Sources will include police-reported crashes, 311 hazards, bicycle facilities, and OSM network extracts.
- We will normalize schemas, enforce CRS, de-duplicate, and write outputs as **GeoParquet** and vector tiles.
- The risk surface will be built from multi-scale **hexbin/KDE** with **time-of-day bins** (Morning/Midday/Evening/Night).

Cloud Services (AWS) — why we will use these four

1. **Amazon S3 (Data lake & tiles).** We will use S3 as the single, durable place to store our raw and clean datasets, along with risk tiles. It's cheap, easy to share online, and both the frontend and backend can access it.
2. **AWS Lambda (Routing API).** We will place API Gateway in front of Lambda so the browser has one simple, secure URL to access. It provides built-in HTTPS, request limits, and clear versioned endpoints.
3. **Amazon API Gateway (HTTPS entry point).** We will put API Gateway in front of Lambda so the browser has one simple, secure URL to call. It gives us built-in HTTPS, request limits, and clean versioned endpoints.
4. **Amazon RDS for PostgreSQL (+ PostGIS).** We will keep the street network and risk summaries in RDS because it is a dependable database with spatial features (PostGIS) that allow us to perform quick map queries and maintain data consistency.

Helpful extras (not required for the "four"): **AWS Glue** + **EventBridge** to refresh data on a schedule (ETL), **CloudFront** to make map tiles and the site load faster, **CloudWatch** to see logs and set alerts, **Secrets Manager** to keep keys and passwords safe, and (optional) **ElastiCache/Redis** to cache recent routes for speed.

Hardware

- No dedicated hardware will be provisioned; all compute and storage will be managed cloud services.

Architectural Diagram (interactions between components)

The diagram below will serve as the implementation blueprint.

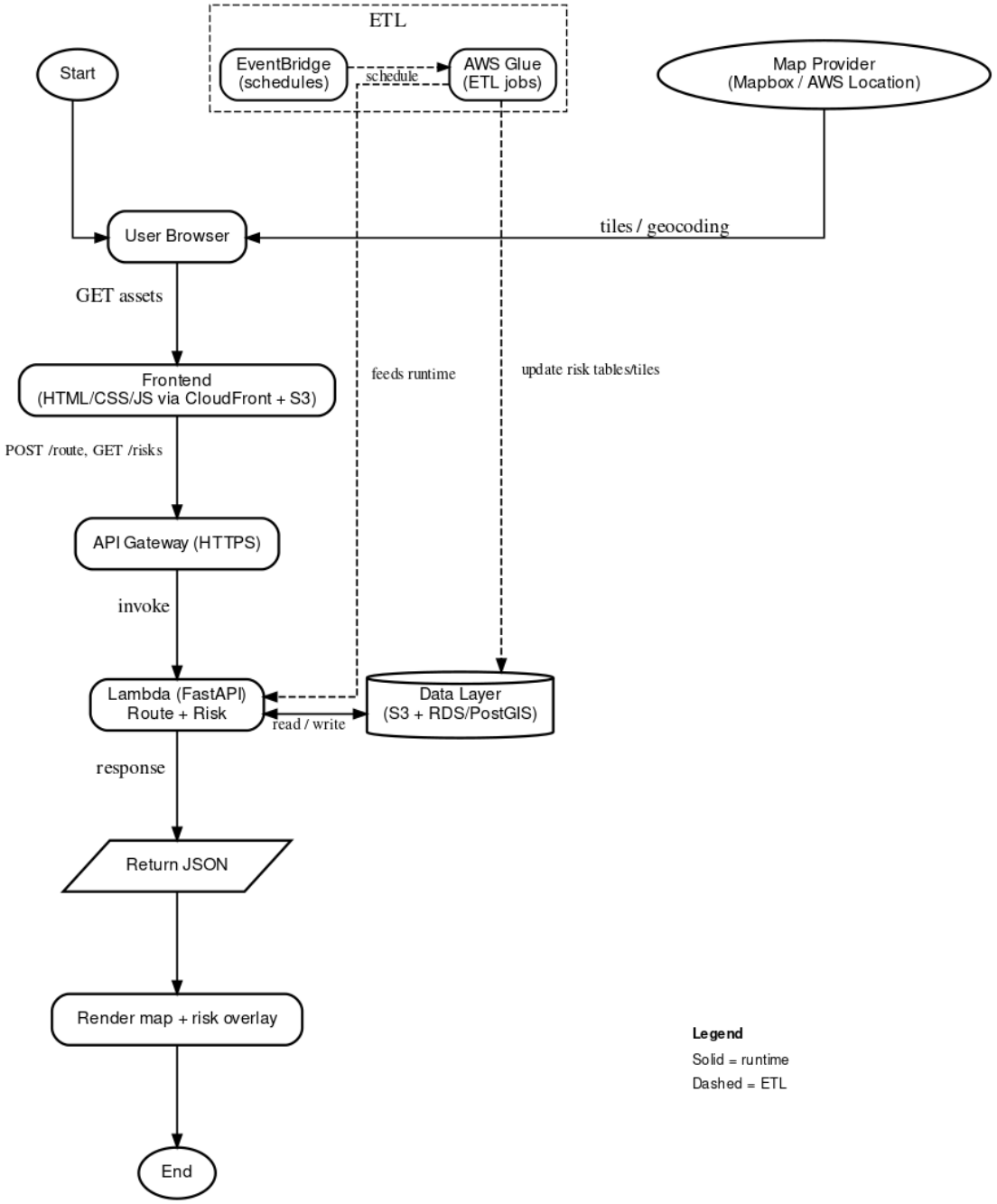

Figure 1: Safer Ride — AWS overview. Solid = runtime; dashed = ETL.

Interaction of the Software/Hardware Components

1. At runtime, the browser will load static assets from **CloudFront+S3** and users will enter origin/destination (debounced geocoding).

2. The browser will call `/route` on **API Gateway**, which will invoke **Lambda (FastAPI)**.

3. Lambda will compute **Fastest** and **Safer** polylines. Risk values will be retrieved from **RDS/PostGIS** and/or sampled from tiles stored in **S3**.

4. The API will return polylines and metrics; the browser will render routes, markers, and the risk overlay with legend.

5. On a schedule, **EventBridge→Glue** will refresh datasets, recompute risk summaries/tiles, and write to **S3/RDS**.

Debugging and Testing Mechanisms (required "Debug" rubric)

Unit tests (PyTest) will cover ETL transforms (schema/CRS, nulls, bounds), risk normalization to $[0, 1]$, and routing cost-function behavior.
Integration tests will validate golden Denver OD pairs with expected time/risk deltas, contract tests for `/route` JSON, and tile availability for `/risks`.
UI tests will exercise theme toggles (layer persistence), keyboard navigation/ARIA, and marker visibility/z-index.
Performance tests (Locust) will drive API Gateway→Lambda at 50–200 RPS bursts and report p50/p95 latency and cache-hit rate.
Observability will rely on CloudWatch logs/metrics/alarms; optional X-Ray tracing will be enabled for slow paths.

Why This Will Meet the "Use of Four Cloud Technologies" Requirement

This project will use **four AWS services** in production paths: **Amazon S3**, **AWS Lambda**, **Amazon API Gateway**, and **Amazon RDS (PostgreSQL + PostGIS)**.

1. **Amazon S3** — data lake (raw/clean GeoParquet) and hosting of public risk tiles/static assets.

2. **AWS Lambda** — serverless compute that will implement routing and risk aggregation.

3. **Amazon API Gateway** — managed HTTPS API that will front the routing service.

4. **Amazon RDS (PostgreSQL + PostGIS)** — spatial database that will power edge-level risk queries and summaries.

These services map directly to runtime and ETL flows in the architectural diagram.

Compelling, Scope, and Ambition

Compelling. The project will address a real safety need (sparked by a recent crash in our circle), combining geospatial analysis with routing to produce a tool cyclists can use immediately.
Scope & ambition. We will keep the MVP focused on Denver, a simple yet defensible risk objective, and a minimal UI—ambitious but achievable within the course timeline. If progress is ahead of schedule, we will add stretch items listed below.

Future Implementations (beyond MVP)

- **Temporal risk modeling:** we will explore hour-of-week risk curves and seasonal baselines.
- **Personalization:** riders will be able to set risk tolerance (β) and preferred facility types.
- **Crowd signals:** we will prototype user-flagged hazards with lightweight moderation.
- **A/B evaluation:** we will compare uptake when the safer route is emphasized vs. presented as an alternative.