

Latency Topology Visualizer

Documentation and Instructions to Run

Yashpal Yadav

<https://github.com/Yashpal-0/latency-topology-visualizer>

Quick Run Instructions

1. Ensure `Node.js` 18+ and `npm` are installed.
2. Copy the provided `.env.local` file to project root location.
3. Install dependencies: `npm install`
4. Start the development server: `npm run dev`
5. For production build: `npm run build` then `npm run start`

1 Assignment Summary

1.1 Objective

Design and implement a Next.js TypeScript application that renders a 3D world map to visualise:

- Major cryptocurrency exchange server locations
- Real-time network latency between cloud co-location regions (AWS, GCP, Azure) and exchanges
- Historical latency trends with statistical summarisation

1.2 Key Functional Requirements

- F1** Interactive 3D globe with smooth camera controls using a modern mapping library
- F2** Distinct exchange markers with metadata on hover/click, provider-based legend
- F3** Animated, colour-coded real-time latency connections refreshed every 5–10 seconds
- F4** Historical latency chart with range selector (1h, 24h, 7d, 30d) and descriptive statistics
- F5** Cloud provider region visualisation with filters and server information overlays
- F6** Control panel for search, filtering, layer toggles, and performance metrics
- F7** Responsive and touch-friendly design suitable for desktop and mobile contexts

1.3 Technical Requirements Overview

- TypeScript across the codebase, leveraging React Hooks and the Next.js App Router
- Real-time data sourced from Cloudflare Radar (free tier) via custom API routes
- Proper error handling, caching, and loading states
- Performance-conscious rendering and animation strategies
- Documentation, video walkthrough, and reproducible setup instructions

2 Solution Architecture

2.1 High-Level Overview

- **Frontend (Client Components):** Built on Next.js with the App Router. The root page instantiates a single `ExchangeMap` component, which orchestrates Mapbox GL for the 3D globe, overlays, control panel, and the historical latency panel.
- **API Layer (Server Components):** Located under `src/app/api`, exposing:
 - `/api/latency`: returns real-time snapshots derived from Cloudflare Radar.
 - `/api/latency/history`: returns historical timeseries data and summary statistics.
- **Static Data and Types:**
 - `src/data/network.ts`: curated exchange and cloud region metadata.
 - `src/types/latency.ts`: shared TypeScript contracts for latency data structures.
- **Styling & Theming:** Tailwind CSS (via `@import "tailwindcss"`), supplemented by custom CSS in `globals.css` for marker aesthetics and dark theme ambience.

The application follows a declarative, component-driven architecture, with React state managing UI selections and Mapbox layers updated via imperative APIs to avoid unnecessary re-mounts.

3 Project Structure

```
src/  
  app/  
    layout.tsx      # Global HTML shell, fonts, metadata  
    page.tsx        # Top-level page rendering ExchangeMap  
    globals.css     # Tailwind base + custom CSS for markers/popups  
    api/  
      latency/  
        route.ts    # Real-time snapshot endpoint  
      latency/history/  
        route.ts    # Historical latency endpoint  
  components/  
    ExchangeMap.tsx  # Main client component (map, overlays, controls)  
    LatencyHistoryPanel.tsx  
                      # Recharts line chart subcomponent  
  data/  
    network.ts      # Static lists for exchanges and cloud regions  
  types/  
    latency.ts      # Shared TypeScript types for latency structures
```

4 Key Technologies

- **Next.js 13+ App Router:** Server-side API routes and client components.
- **TypeScript:** Type safety across components, APIs, and data processing.
- **Mapbox GL JS:** Globe projection, geographic layers, custom markers, animation.
- **Cloudflare Radar IQI API:** Public latency metrics.
- **Recharts:** Responsive timeseries chart for historical latency.
- **Tailwind CSS + Custom CSS:** Layout, typography, and marker styling.

5 Static Data Definitions

5.1 Exchange Catalogue

Located in `src/data/network.ts`:

- An array of `ExchangeLocation` objects with fields: `id`, `name`, `city`, `country`, `provider` ("AWS" | "GCP" | "Azure"), and geographic coordinates.
- This dataset drives:
 - Mapbox markers and layer metadata
 - Control panel dropdowns
 - Search and filter features
- Example snippet:

```
1 {  
2   id: 'binance-ldn',  
3   name: 'Binance',  
4   city: 'London',  
5   country: 'United Kingdom',  
6   provider: 'AWS',  
7   coordinates: [-0.1276, 51.5072],  
8 }
```

5.2 Cloud Regions Catalogue

Also in `src/data/network.ts`:

- Array of `CloudRegion` entries representing AWS, GCP, and Azure co-location regions.
- Fields include: `name`, `regionCode`, `serverCount`, `coordinates`.
- Consumed by:
 - Mapbox markers and polygon overlays
 - API endpoints for ID validation
 - Control panel filters and region details card

5.3 Shared Types

Defined in `src/types/latency.ts`:

- `TimeRangeKey`: union of allowed ranges
- `LatencyHistoryPoint`: standardised structure for a single historical sample
- `HistoryStats`: container for min/max/avg/samples metadata
- These types are imported across client and server code, guaranteeing consistent interfaces.

6 API Layer

6.1 /api/latency (Real-Time Snapshots)

Path `src/app/api/latency/route.ts`

Responsibilities

- Accepts optional `regions` query param (comma-separated region IDs) and `range`.
- Validates the presence of `CLOUDFLARE_API_TOKEN`; returns 500 otherwise.

- Utilises an in-memory cache (TTL: 10 seconds) keyed by the set of region IDs and the range.
- For cache misses, delegates to the history endpoint to fetch the latest sample per region.
- Returns a concise payload:

```

1 {
2   data: Array<{
3     regionId: string;
4     location: string;
5     latencyIdle: number | null;
6     latencyLoaded: number | null;
7     jitterIdle: number | null;
8     capturedAt: string;
9   }>;
10  cachedAt: string;
11  cacheTtlMs: number;
12 }

```

Error Handling

- Missing token → 500
- Unknown region IDs → 400
- Downstream fetch failures produce descriptive error logs and degrade gracefully.

Caching Strategy

- Map<string, CacheEntry> where each entry stores `data` and `fetchAt`.
- Prevents hammering Cloudflare's API while still providing a near-real-time experience.

6.2 /api/latency/history (Historical Data)

Path `src/app/api/latency/history/route.ts`

Responsibilities

- Accepts `region` and `range` query parameters.
- Validates the target region against `CLOUD_REGIONS`.
- Translates the range key to a duration (e.g. "24h" → 24 hours).
- Builds a Cloudflare Radar IQI API request, buffering the end time by 60 seconds (per API requirement).
- Parses varied response formats (series, histograms, percentile series) into a canonical array of `LatencyHistoryPoint`.
- Computes statistics via `computeHistoryStats`.
- Returns:

```

1 {
2   points: LatencyHistoryPoint[];
3   stats: HistoryStats | null;
4   queriedAt: string;
5 }

```

Resilience

- Reflective logging for debugging (e.g. evidence of external errors).
- Graceful handling of missing fields, fallback to empty arrays, and null statistics.
- Strict typing and helper functions to unify the response shape irrespective of Cloudflare's payload variant.

7 Client-Side Visualisation

7.1 ExchangeMap Component

File `src/components/ExchangeMap.tsx`

Mapbox GL Setup

- Waits for `containerRef` and `NEXT_PUBLIC_MAPBOX_ACCESS_TOKEN`.
- Configures the globe projection (`projection: 'globe'`) with pitch, bearing, and zoom suited to a global overview.
- Adds navigation, fullscreen, and scale controls for user interaction.
- Enhances visuals by enabling fog, terrain, and atmospheric sky when the style loads.
- Registers custom sources/layers:
 - Region boundary polygons (`fill` and `line` layers).
 - Latency connections (`line` layer) and label overlay (`symbol` layer).

State Management

- React `useState` for:
 - Live latency snapshots
 - Provider visibility toggles
 - Layer toggles (real-time lines, history panel, region boundaries)
 - Exchange filter, search query, and selection state
 - Historical chart data, stats, loading and error flags
- `useRef` mirrors to synchronise with external Mapbox callbacks (avoids stale closures).

Marker Rendering

- Exchanges: circular markers with glow using custom CSS (`.exchange-marker`). Hover and click interactions show popups and animate camera transitions via `map.flyTo`.
- Cloud Regions: diamond markers (`.cloud-region-marker`). Display additional metadata (region code, server count) in popups.
- All markers stored in refs to toggle visibility without remounting.

Latency Connections

- Generated via helper `toFeatureCollection`, mapping exchange-region pairs to `LineString` features.
- Latency value determines colour (green/yellow/red/grey) using `getLatencyStatus`.
- Animated using `line-dasharray` updates on a timer to simulate pulses.
- Hover popups show exchange name, region, provider, server count, and current latency.

Polling Loop

- Effect hook triggers an immediate fetch to `/api/latency`, then every 10 seconds.
- Updates state with the latest snapshots; handles errors by presenting messaging in the UI.
- Aborts on component unmount to prevent memory leaks.

Layer Visibility

- Toggling `layerVisibility.realtime` hides/shows the Mapbox line and symbol layers.
- Toggling `layerVisibility.regions` hides/shows polygon fill and outline layers, and loops through markers to toggle their DOM visibility.
- Toggling `layerVisibility.history` controls whether the historical panel is rendered, and clears data when disabled.

Control Panel and Overlays

- **Left Column:** introduction card, last update time, error banner, search input with results list, exchange filter dropdown, cloud provider toggles, latency range pills, and layer toggles.
- **Right Column:** performance snapshot showing overall status, sample counts, visible regions, and min/avg/max metrics; latency legend clarifying colour thresholds.
- **Bottom Centre:** region details card summarising provider, region code, server count, and city/country for the active region.
- **Bottom Right:** conditional rendering of `LatencyHistoryPanel` when history layer is active.
- **Missing Token Overlay:** if `NEXT_PUBLIC_MAPBOX_ACCESS_TOKEN` is absent, a blocking message instructs the user to configure it.

Responsiveness

- Overlays use flexible widths and rely on Tailwind utilities to adapt to viewport changes.
- Mapbox GL natively supports touch gestures, enabling mobile pan/zoom/rotate without extra code.
- `handleResize` writes viewport dimensions to CSS variables, ready for advanced responsive styling needs.

7.2 LatencyHistoryPanel Component

File `src/components/LatencyHistoryPanel.tsx`

Purpose

- Present historical latency for the active exchange-region pair.
- Offer selectors for exchange, region, and time range.
- Display min/avg/max statistics and a responsive Recharts line chart.

Key Features

- Accepts props from `ExchangeMap` (options, selections, data, status).
- Formats latency values to two decimal places with units.
- Handles loading, error, and empty states within the chart container.
- Uses `ResponsiveContainer` so the chart fills its parent area seamlessly.
- Axis labels and tooltips use locale-aware formatting for timestamps.

8 Data Flow and State Synchronisation

8.1 Real-Time Polling Loop

1. Component mounts → fetch `/api/latency` with all region IDs.
2. API returns snapshots → convert to `Record<regionId, LatencySnapshot>` for $O(1)$ lookup.
3. Update Mapbox GeoJSON source with `toFeatureCollection` respecting provider/latency filters.
4. Set a 10-second interval to repeat steps 1–3.
5. On unmount, clear interval and abort outstanding fetch.

8.2 Historical Fetch Cycle

Triggered whenever `selectedExchange`, `selectedRegion`, `selectedRange`, or history layer visibility changes.

1. Effect starts, sets loading true, clears previous error.
2. Builds query string with `region` and `range`.
3. Calls `/api/latency/history`, passing an `AbortController` signal.
4. On success, updates `historyPoints`, `historyStats`, and `historyQueriedAt`.
5. On failure, sets descriptive error message.
6. Ensures loading flag is reset unless the request was aborted.

9 Filtering, Search, and Layer Logic

9.1 Provider Visibility

Stored in `visibleProviders`, e.g.

```
1 { AWS: true, GCP: true, Azure: true }
```

- Toggling updates React state and a ref mirror.
- Mapbox boundary source is regenerated with only visible providers.
- Region markers respected by toggling DOM visibility.
- Performance metrics recompute visible region count accordingly.

9.2 Latency Bands

- Buttons control a `LatencyVisibilityMap`, e.g.:

```
1 { low: true, medium: true, high: true, unknown: true }
```

- Prevents all bands from disabling simultaneously (maintains at least one active status).
- `toFeatureCollection` only emits features whose status is enabled.

9.3 Exchange Filter

- Dropdown lets users pick a specific exchange or view all.
- Filter affects which exchange-region lines appear on the globe.
- Search selection also updates this value and flies to the chosen exchange.

9.4 Search

- Filters both exchanges and regions by name, city, or ID.
- Results show the provider colour indicator; disabled if that provider is currently hidden.
- Selecting a search result updates the relevant selection and flies the camera to that location.

10 Styling and Visual Design

10.1 Typography and Layout

- Next.js `metadata` currently uses the default title/description and could be updated to project-specific copy.
- Root layout loads Geist Sans and Geist Mono for consistent typography.
- Tailwind utility classes create consistent spacing, rounding, and colour palette aligned with a dark trading-dashboard aesthetic.

10.2 Custom CSS Highlights (`globals.css`)

- Marker shapes (`.exchange-marker`, `.cloud-region-marker`) with glow and pseudo-elements for stems/borders.
- Popup styling (`.marker-popup`, `.latency-popup`) for legible, branded tooltips.
- Latency chips (`.latency-low`, etc.) providing colour-coded labels reused across line pop-ups and legend.
- Body defaults fallback to system sans-serif if custom fonts fail.

11 Performance Considerations

- Real-time polling is throttled server-side via caching and client-side via a 10-second interval.
- Mapbox GL layers use `setData` and layout/paint property updates rather than reconstructing the map instance.
- Dash-array animation uses lightweight `setInterval`; cleared on unmount to avoid resource leaks.
- Cloudflare fetches use `no-store` caching directive per API guidelines; errors produce concise fallback states.
- Recharts disables animation for the line chart to reduce CPU load on frequent updates.

12 Setup and Deployment

12.1 Prerequisites

- Node.js 18+
- npm
- Mapbox account for access token
- Cloudflare API token (Radar IQI read access)

12.2 Environment Variables

`.env.local` has been provided with the submission:

12.3 Installation and Running

```
npm install
npm run dev
```

Open the provided localhost URL (typically `http://localhost:3000`). Ensure the browser console shows no missing-token messages.

12.4 Build

```
npm run build
npm run start
```

Produces an optimised production build using Next.js.

13 Testing and Validation

- Manual validation of all assignment requirements (map interaction, marker popups, latency animation, history panel toggles).
- API error scenarios tested by removing tokens and verifying user-facing messages.
- Responsiveness verified via browser device emulation (mobile/tablet).
- Mapbox token overlay confirmed to prevent blank screen when misconfigured.
- Future work: automated tests could be added for util functions (e.g. latency parsing) using Jest or Vitest.

14 Assumptions & Limitations

- Cloudflare Radar provides region-level latency to ISP territories; exchange-to-region latency is approximated by linking known exchange locations to nearest co-location regions.
- No persistent datastore is used—real-time and historical data are fetched on demand.
- API rate limiting relies on Cloudflare’s public quota; additional caching or scheduling may be required for large scale deployment.
- Mapbox GL requires WebGL; older devices/browsers may fall back to limited performance.
- No heatmap overlay implemented; architecture allows easy extension by adding a new Mapbox source/layer.

15 Potential Enhancements

- Introduce great-circle arcs to better represent long-distance latency paths.
- Layer in a latency heatmap by sampling points on the globe and colour coding via Mapbox raster layers.
- Support dark/light theme toggling by extending Tailwind configuration and marker styling.
- Persist user preferences (filters, toggles) via local storage or server state.
- Expand data source to include specific exchange latency endpoints if available.
- Incorporate trading volume or order flow animations for additional context.