Smart Commute & Air Quality Advisor

A Real-Time Decision Support System for Urban Commuters

Author: Chintada Bargav Sree Naidu

September 7, 2025

Project Repository: GitHub

Abstract

This report documents the design, development, and deployment of the "Smart Commute & Air Quality Advisor," a web application created to address the complex daily transit decisions faced by commuters in metropolitan areas. The system provides users with a real-time, actionable recommendation for their commute by aggregating and synthesizing data from three distinct external services: Google Maps for traffic and transit information, the World Air Quality Index (WAQI) project for air pollution data, and the OpenWeatherMap One Call API for detailed weather forecasts.

The application is built in Python using the Streamlit framework for the user interface. The backend orchestrates multiple API calls, handles data processing, and implements a rule-based recommendation engine. For deployment, the application was containerized using Docker and deployed on AWS Elastic Container Service (ECS) with Fargate, demonstrating a complete end-to-end, cloud-native workflow. This report details the system architecture, the iterative development process, the technical challenges encountered during deployment, and the solutions implemented to create a robust and user-friendly tool.

Contents

1 Introduction

Daily commuters in large Indian cities like Kolkata face a multifaceted challenge. The decision of how to travel from home to work is not merely a question of distance, but a complex trade-off between traffic congestion, adverse weather conditions, and significant health risks associated with poor air quality. Information to make an informed decision is typically fragmented across multiple applications—a mapping service for traffic, a weather app, and a separate environmental portal for air quality.

This project was undertaken to solve this problem by creating a single, unified dashboard. The primary goal was to build a system that could:

- Consolidate real-time data for traffic, weather, and air quality for a user's specific location.
- Synthesize this disparate data into a single, easy-to-understand, and actionable recommendation.
- Provide a seamless user experience with advanced input methods like GPS and interactive maps.
- Follow modern software engineering best practices, including containerization and cloud deployment.

The result is a practical tool that empowers users to make faster, healthier, and more informed decisions about their daily commute.

2 System Architecture and Technology Stack

The application is designed as a modern, I/O-bound microservice relying on external APIs for its data.

2.1 Technology Stack

- **Frontend & Application Logic:** Python 3.9, Streamlit
- **Data Handling:** Pandas
- **Interactive Maps:** Folium, streamlit-folium
- **Geolocation:** streamlit-geolocation
- **Containerization:** Docker
- **Cloud Deployment:** AWS ECS (Fargate), Docker Hub, AWS IAM, AWS Secrets Manager

2.2 Data Sources (APIs)

The core intelligence of the application is derived from the real-time data provided by three external APIs:

- **Google Maps Directions API:** Used to fetch driving and public transit durations. The `departure_time="now"` parameter was used to obtain predictive traffic data, allowing for the calculation of real-time delays.
- **World Air Quality Index (WAQI) API:** Provides real-time Air Quality Index (AQI) data and identifies the dominant pollutant. The API supports queries by both city name and geographic coordinates.
- **OpenWeatherMap One Call API 3.0:** Provides weather data from a single call, including current conditions, temperature, alerts, and detailed hourly forecasts. It requires geographic coordinates, necessitating a preliminary geocoding step.

2.3 Recommendation Engine

The "brain" of the application is a rule-based decision engine implemented in Python. It does not use a machine learning model but rather a prioritized decision tree:

1. **Adverse weather:** If conditions like rain or thunderstorms are present, always recommend driving for safety and comfort.
2. **Traffic:** If weather is clear, analyze the traffic delay. Significant delay (e.g., ¿20 minutes) triggers a public transit recommendation.
3. **Air Quality:** If traffic and transit times are comparable and weather is clear, a high AQI (¿150) recommends driving with windows closed.

3 Deployment Workflow

The application was deployed using a standard DevOps pipeline:

1. **Containerization:** The final Python application with dependencies from `requirements.txt` was packaged into a Docker image using a multi-stage Dockerfile based on `python:3.9-slim`.
2. **Image Registry:** The Docker image was pushed to a public Docker Hub repository for cloud access.
3. **Cloud Deployment on AWS ECS:**
   - ECS cluster using **AWS Fargate**.
   - Secrets managed with **AWS Secrets Manager**.
   - Dedicated **IAM Task Execution Role** with appropriate `secretsmanager:GetSecretValue` permissions.
   - ECS Task Definition referencing Docker Hub image; secrets injected as environment variables.
   - Service to run and maintain one task, with a security group allowing traffic on port 8501.

4 Challenges and Solutions

Deployment revealed real-world technical challenges requiring systematic debugging:

- **Challenge 1: Local vs. Cloud Configuration**—Code was designed to read secrets from a local `secrets.toml` file (`st.secrets`); the containerized cloud setup provides keys as environment variables, so the app crashed.
- **Solution 1:** Refactored code to use `os.environ.get()` for environment variable access, falling back to `st.secrets` if not found. This allows seamless operation locally and in cloud environments.
- **Challenge 2: AWS Permissions and Circuit Breaker**—The "ECS Deployment Circuit Breaker" error indicated container crashes. Task logs showed `AccessDeniedExceptio` when fetching secrets: IAM role with permissions was incorrectly assigned to "Task Role" rather than **Task Execution Role**.
- **Solution 2:** Corrected IAM role assignment in the ECS Task Definition. Once the Execution Role had proper permissions, the issue was resolved.

5 Conclusion

The Smart Commute & Air Quality Advisor was successfully developed and deployed, demonstrating a complete end-to-end cloud-native workflow. Key lessons included handling IAM configuration and practical debugging on AWS ECS. The resulting tool demonstrates robust real-time API integration, user-focused UI engineering, and scalable cloud deployment—essential portfolio skills for modern data and software engineers.