Project Blueprint: AeroStream UAV Telemetry & Command Mesh

Author: Khritish Kumar Behera

Project Goal: To architect a high-frequency, resilient, and scalable backend system for real-time monitoring and command of Unmanned Aerial Vehicles (UAVs).

Executive Summary

AeroStream is a distributed systems project designed to handle high-velocity spatial data from a virtual fleet of 500+ drones. The system addresses common industry challenges: high-throughput data ingestion, multi-protocol communication (REST, gRPC, WebSockets), event-driven decoupling, and cloud-native orchestration.

Week 1: The Flight Data Core (Data Architecture)

Overview: The foundation of the system. We move away from simple "flat" databases to an optimized "Time-Series" architecture designed for high-write loads.

- **Aim:** Create a "Black Box" recorder that can handle concurrent telemetry streams without performance degradation.
- **Tech Stack:** Python (FastAPI), PostgreSQL (Time-Series Partitioning), Pydantic, asyncpg.
- **Deliverables:**
  - **Partitioned Database Schema:** A PostgreSQL instance where tables are split by time-ranges (e.g., daily partitions).
  - **Telemetry Ingestion API:** A RESTful endpoint (*/api/v1/telemetry*) optimized for batch writes.
  - **Drone Fleet Simulator:** A Python script modeling flight physics (climb, cruise, rotate) and sending real-time sensor data.
- **System State:** The system can receive telemetry from 500 virtual drones and store them in a way that allows for instant retrieval of the last 10 minutes of flight logs.
- **Quantifiable Metrics:**
  - "Architected a PostgreSQL schema using **Time-Series Partitioning**, enabling efficient storage of **1M+ flight data points** per hour."
  - "Developed a high-concurrency API handling **2,000+ Requests Per Second (RPS)** with a p99 latency of <80ms."

Week 2: Advanced Networking & Observability (Multi-Protocol Layer)

Overview: Real-world systems don’t just use REST. Internal services need speed, and dashboards need specific data.

- **Aim:** Optimize network communication and implement real-time alerting for critical flight conditions (e.g., low battery, unauthorized altitude).
- **Tech Stack:** gRPC (Protobuf), GraphQL (Strawberry/Ariadne), WebSockets, Grafana.
- **Deliverables:**
    - **Internal gRPC Service:** An internal "Aggregator" service that uses binary Protocol Buffers for 10x faster service-to-service communication.
    - **GraphQL Query Layer:** An interface for front-end tools to request specific telemetry fields (e.g., *only* altitude and GPS).
    - **WebSocket Alerting:** A push notification channel that triggers when a simulated drone enters a "Warning" state.
- **System State:** You can "see" the flight data in a Grafana dashboard, and the system sends a live alert to the console if a drone's battery drops below 15%.
- **Quantifiable Metrics:**
    - "Reduced internal network payload size by **65%** by migrating service communication from JSON/REST to **gRPC/Protobuf**."
    - "Implemented real-time spatial alerting via **WebSockets**, achieving a **sub-50ms latency** from event detection to notification."

Week 3: Resilient Messaging & Scalability (Event-Driven Architecture)

Overview: Systems crash during spikes. This week introduces a "Buffer" to ensure the system never loses a single byte of flight data.

- **Aim:** Decouple the Ingestion API from the Database to handle massive traffic spikes and ensure fault tolerance.
- **Tech Stack:** Apache Kafka, Background Workers (Python Consumers).
- **Deliverables:**
    - **Kafka Message Backbone:** An event stream where the API "produces" telemetry and workers "consume" it.
    - **Idempotent Workers:** Consumer logic that ensures data is only written once to the database, even if a service restarts.
    - **Dead Letter Queue (DLQ):** A secondary Kafka topic to catch and debug malformed drone data.
- **System State:** If the Database goes down for 5 minutes, the Drones keep sending data. Once the DB comes back up, the Kafka workers catch up automatically without data loss.

- **Quantifiable Metrics:**
    - "Engineered an event-driven pipeline using **Apache Kafka**, increasing system resilience to handle **4x baseline traffic spikes** during simulated drone swarms."
    - "Eliminated data duplication and loss by implementing **idempotent consumer logic** and a **Dead Letter Queue (DLQ)** for fault-tolerant processing."

Week 4: Production Infrastructure (Cloud-Native & DevOps)

Overview: The final step to prove you are industry-ready. We move the code from "your laptop" to a professional "Production Cluster."

- **Aim:** Standardize the environment and automate the scaling of the system based on the number of active drones.
- **Tech Stack:** Docker, Kubernetes (K8s), Google Cloud Platform (GCP).
- **Deliverables:**
    - **Microservice Containerization:** Multi-stage Dockerfiles for the API, Consumers, and DB.
    - **Kubernetes Manifests:** Configuration files for Deployments, Services, and **Horizontal Pod Autoscaling (HPA)**.
    - **Cloud Deployment:** The entire AeroStream stack running on Google Kubernetes Engine (GKE).
- **System State:** The system is live on a URL. If you increase the simulator from 500 to 5,000 drones, Kubernetes automatically spins up new "API" containers to handle the load.
- **Quantifiable Metrics:**
    - "Orchestrated a 5-service microservice stack using **Kubernetes**, reducing local setup time from 1 hour to **3 minutes via Docker Compose**."
    - "Deployed to **GCP (GKE)** with **Horizontal Pod Autoscaling (HPA)**, maintaining 99.9% availability during simulated high-load stress tests."

Final Project Impact (For Resume Summary)

AeroStream UAV Mesh: Built a distributed telemetry platform handling 5k+ high-frequency data points per second. Leveraged **Kafka** for event-driven resiliency, **gRPC** for low-latency networking, and **Kubernetes on GCP** for cloud-native scaling. Optimized **PostgreSQL** storage for time-series spatial data, reducing query latency by 60%.