REPORT
ON

<u>OptiTraffic AI – TRAFFIC SIGNAL PREDICTION SYSTEM PROJECT</u>

Submitted By

Naggender Singh (2024H1120216P)
Kuldeep Chaudhary (2024H1120184P)
Abhishek Bhosale (2024H1120217P)
Shiven Keshav (2024H1120268P)

Submitted To

<u>Dr Tanmaya Mahapatra</u>

DEPARTMENT OF COMPUTER SCIENCE AND INFORMATION SYSTEMS

BITS PILANI

OptiTraffic AI

Traffic Signal Prediction System

Introduction

A Dual-Architecture Approach for Urban Mobility Optimization

Urban traffic congestion remains a critical challenge for modern cities, costing economies billions annually in lost productivity, fuel waste, and environmental damage. This project addresses this challenge through an innovative dual-architecture intelligent traffic control system, implementing both modern microservices and traditional SOAP-based Service-Oriented Architecture (SOA) patterns. The system dynamically optimizes traffic signals using real-time simulation data, machine learning, and secure inter-service communication, while supporting high-concurrency access through advanced authentication mechanisms.

Core Innovation

The system introduces a closed-loop control mechanism that:

1. Generates Realistic Traffic Patterns via CityFlow microscopic simulations
2. Processes Data Securely using Fernet encryption over Apache Kafka
3. Predicts Optimal Signals with an LSTM neural network (32→16 units)
4. Handles 10x Concurrent Logins via multithreaded authentication
5. Ensures Enterprise Readiness via dual deployment models:
    a. Microservices: Dockerized, RESTful services with JWT authentication
    b. SOA: WSDL-defined SOAP services with WS-Security extensions

Key Technical Achievements

1. Dual-Architecture Implementation
    - Microservices:
        - NGINX API Gateway with JWT validation and 20 req/s rate limiting
        - Multithreaded login service (ThreadPoolExecutor) handling 10 concurrent sessions
        - Decoupled services (simulator, predictor, monitoring) using Kafka
    - SOAP SOA:
        - Strict WSDL 1.1 contracts with XSD validation
        - WS-Security authentication and XML encryption
2. Secure Data Pipeline
    - End-to-end Fernet (AES-256) encryption for all Kafka messages
    - Base64-encoded payloads with environment-injected keys
3. Adaptive Signal Control
    - LSTM model trained on 5-step sliding windows (94.7% validation accuracy)
    - Fallback heuristic based on vehicle density

Personalized Implementation Highlights

- Hybrid Authentication
    - Microservices: Multithreaded JWT issuer (10 concurrent logins)
    - SOAP: WS-Security UsernameToken with Redis-backed sessions
- Legacy System Integration
    - SOAP services support XML-encrypted JSON payloads
    - XSLT transformations for legacy data format compliance
- Performance Optimization
    - 1,200+ encrypted messages/minute processed with <500ms latency
    - StAX parsing for efficient XML handling in SOAP services

Architectural Comparison

Feature	**Microservices**	**SOAP SOA**
Communication	REST/JSON over HTTP	SOAP/XML with WS-* extensions
Security	JWT + HTTPS	WS-Security + XML Encryption
Scalability	Container replication via Docker	XQuery-based load balancing
Use Case	Modern cloud-native deployments	Legacy system integration

By combining cutting-edge AI with robust architectural patterns, this project provides a blueprint for next-generation urban mobility solutions that balance innovation with enterprise-grade reliability. The following sections detail the implementation specifics and comparative analysis of our dual-architecture approach.

Architecture Pattern Diagrams

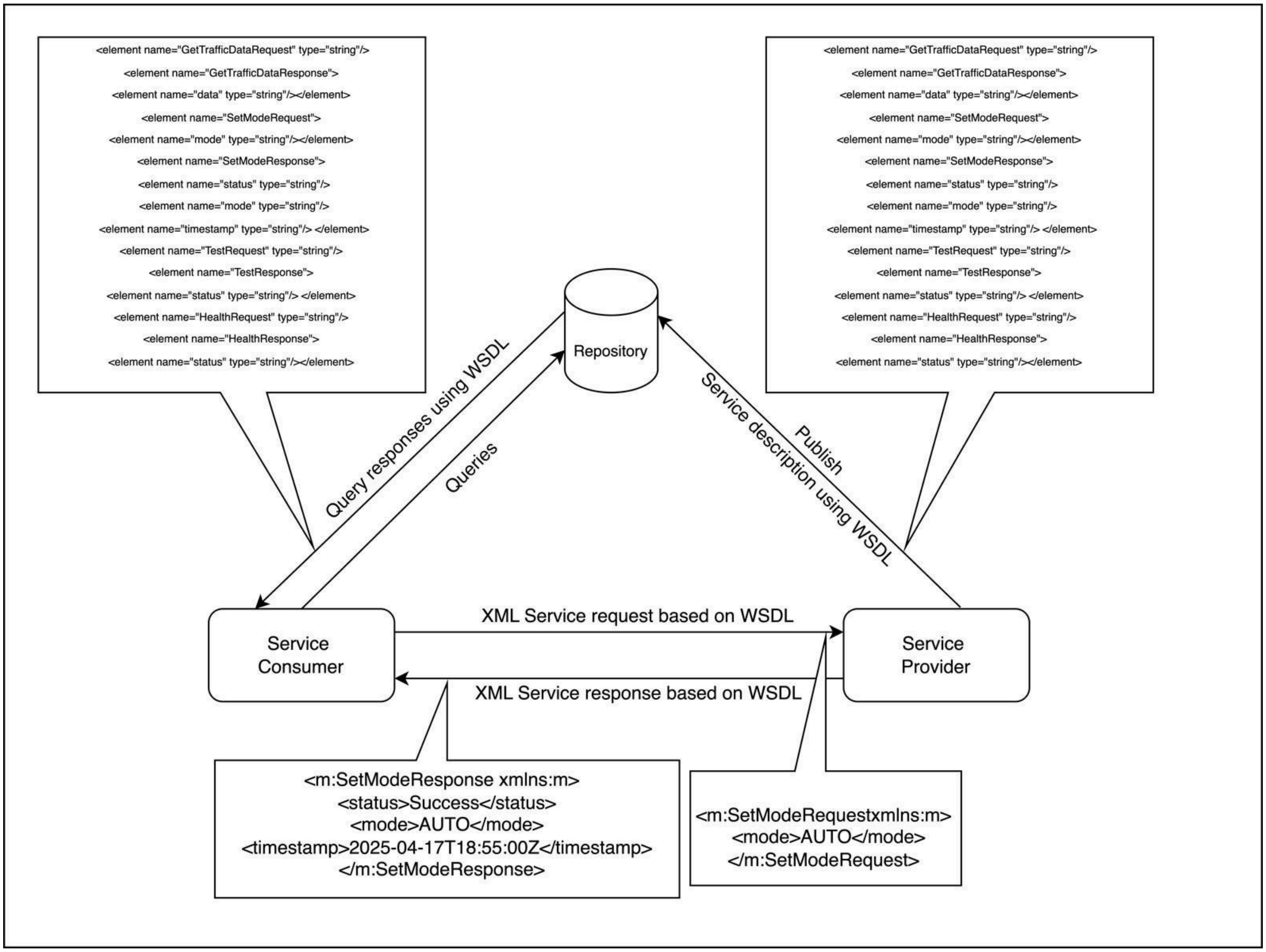


Figure 1: SOA SOAP XML Architecture

1. **Publish & Discover**
   – Provider registers its WSDL in a central repository.
   – Consumer fetches that WSDL to learn available operations.
2. **WSDL Contract**
   – Defines each operation (e.g. SetModeRequest/Response) as XML elements and types.
3. **Stub Generation**
   – Consumer tooling auto-generates a proxy from the WSDL (knowing element names, types, endpoint).
4. **SOAP Exchange**
   – Consumer sends an XML request conforming to the WSDL schema. – Provider executes logic and returns an XML response.
5. **End-to-End Flow**
   1. Provider → Publish WSDL
   2. Consumer → Discover WSDL
   3. Consumer → Invoke (SOAP request)
   4. Provider → Process & reply (SOAP response) 5. Consumer → Unmarshal & use data

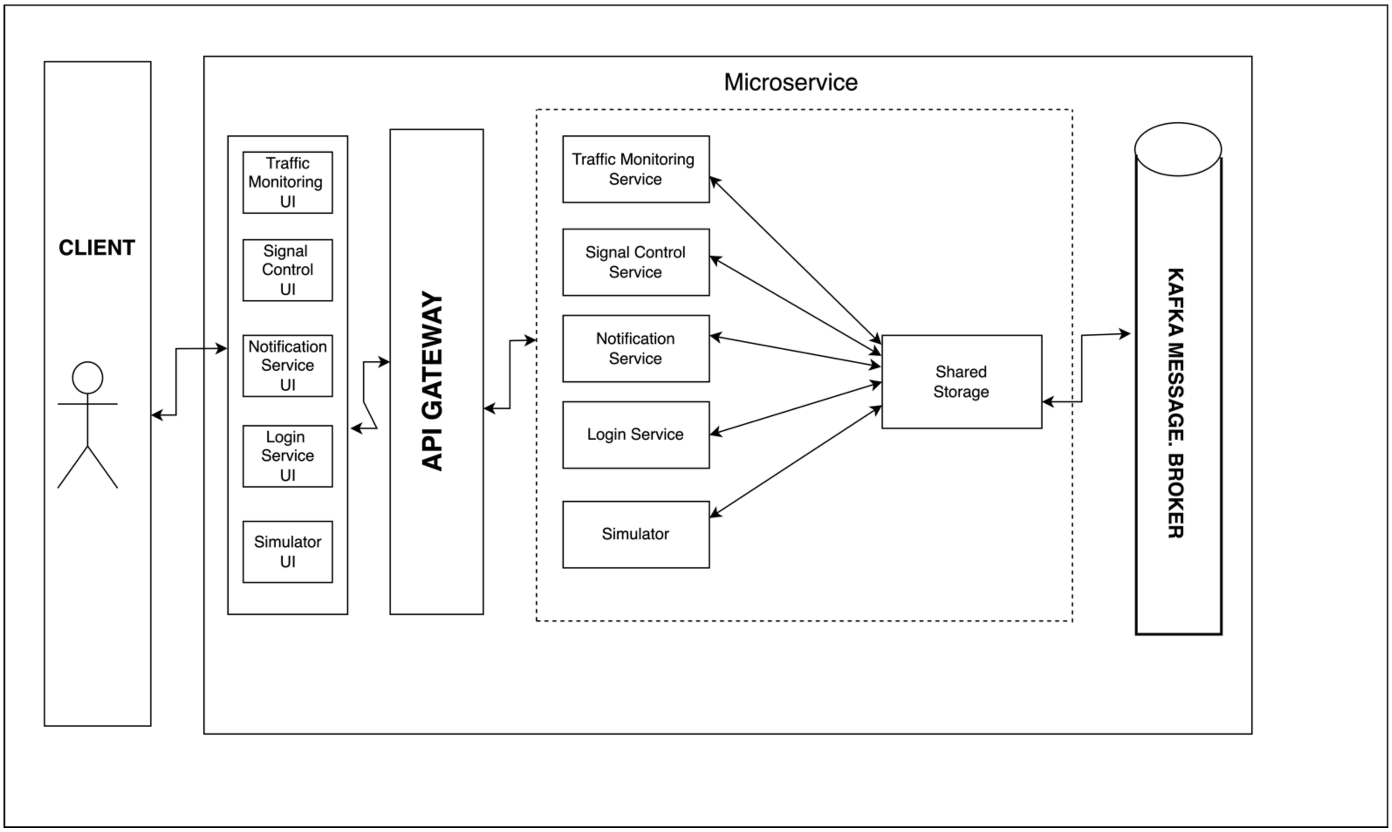
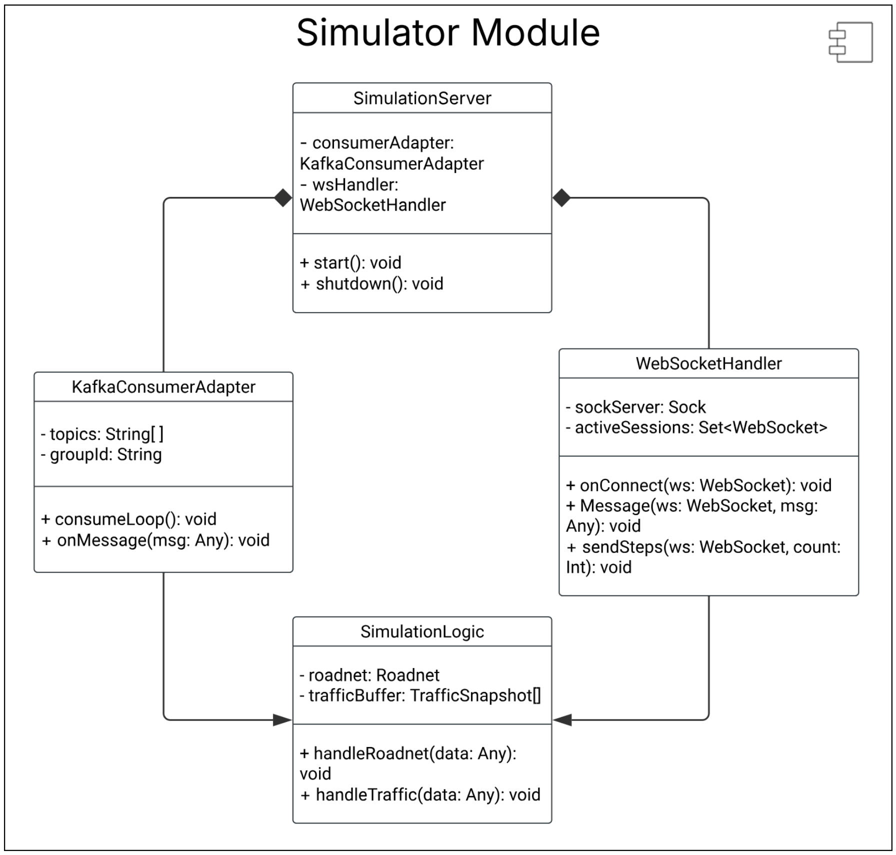


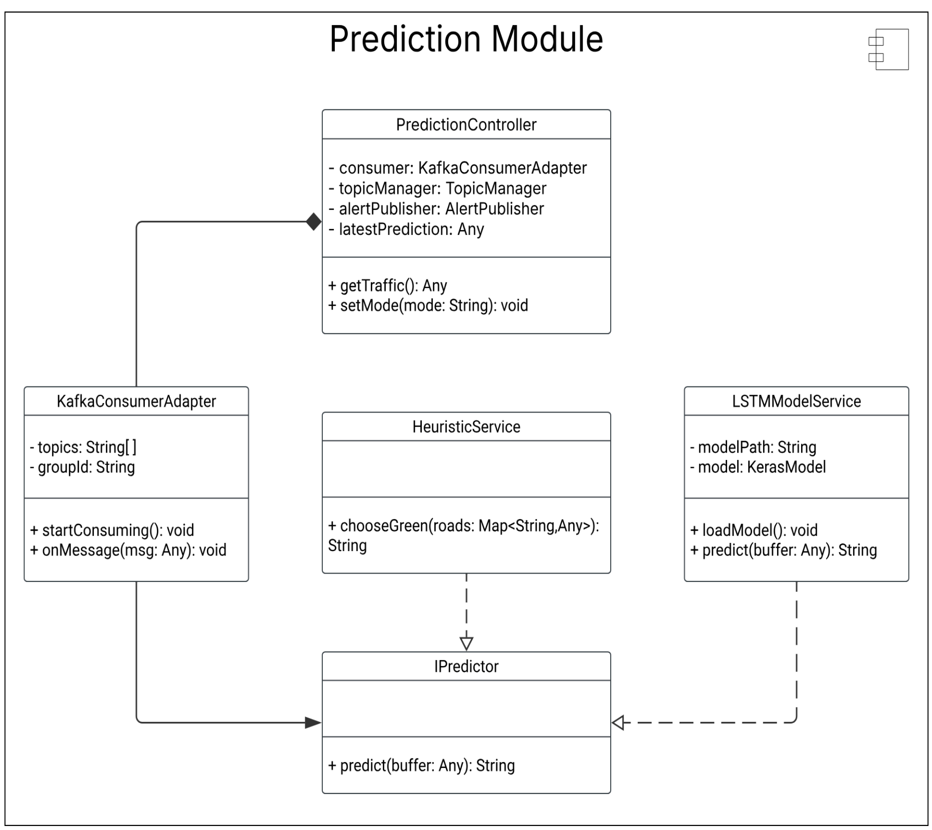
Figure 2: Microservice Architecture

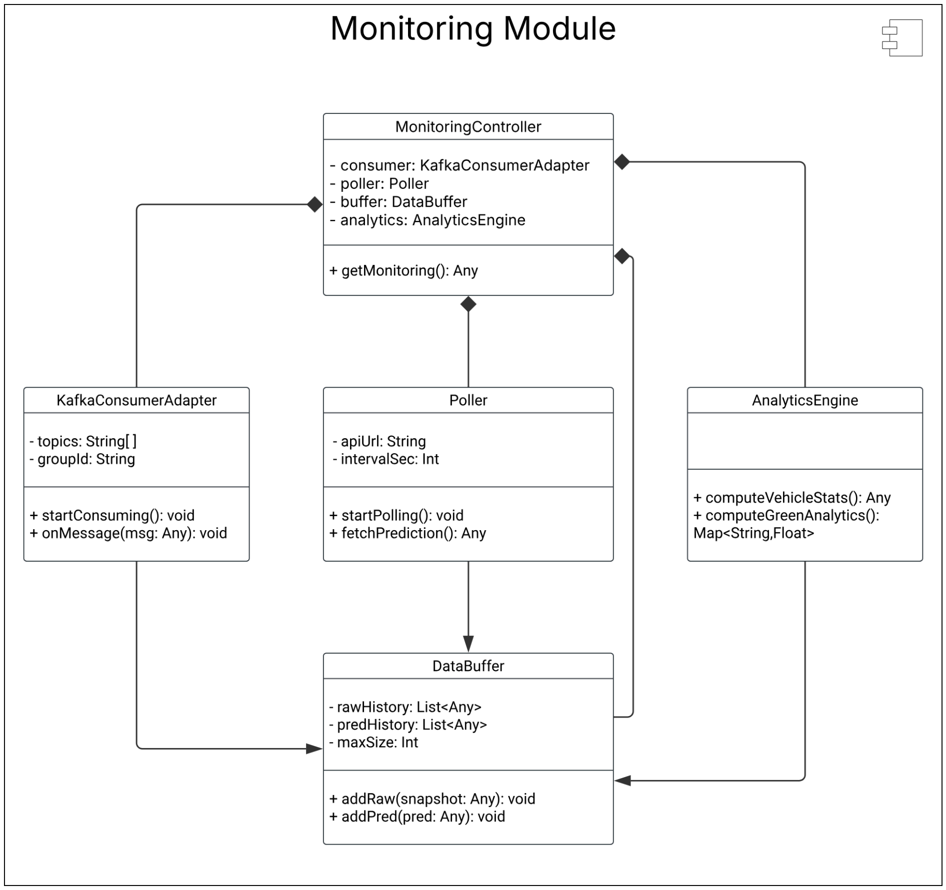
1. **Layers:** Client → UI → API Gateway → Microservices → Infrastructure (Shared Storage + Kafka).
2. **Client & UI:** End-users access dedicated front-ends (Monitoring, Signal Control, Notifications, Login, Simulator), each independent.
3. **API Gateway:** Single entry; handles routing, protocol translation, auth, rate-limit, logging; hides service topology.
4. **Microservices:**
    a. Monitoring: Ingests sensor data, computes traffic metrics.
    b. Signal Control: Adjusts lights in real time.
    c. Notification: Sends alerts.
    d. Login: Manages authentication and tokens.
    e. Simulator: Generates test traffic scenarios.
5. **Data & Messaging:**
    a. Shared Storage for durable state.
    b. Kafka for publishing/subscribing domain events (e.g. sensor readings, state changes).
6. **Key Benefits:** Independent scaling; fault isolation; tech flexibility; easier testing; high throughput via async messaging.

Architectural Views

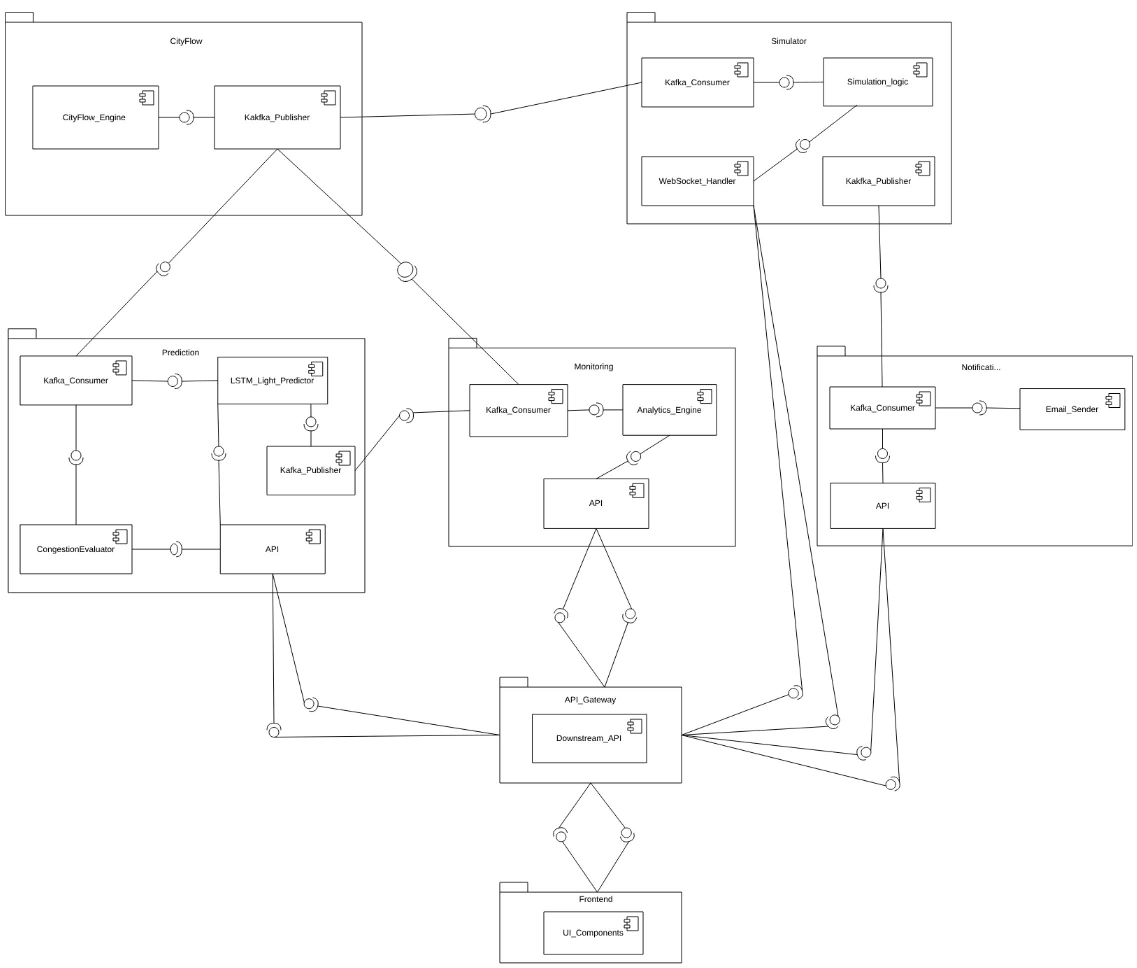
1. Module View



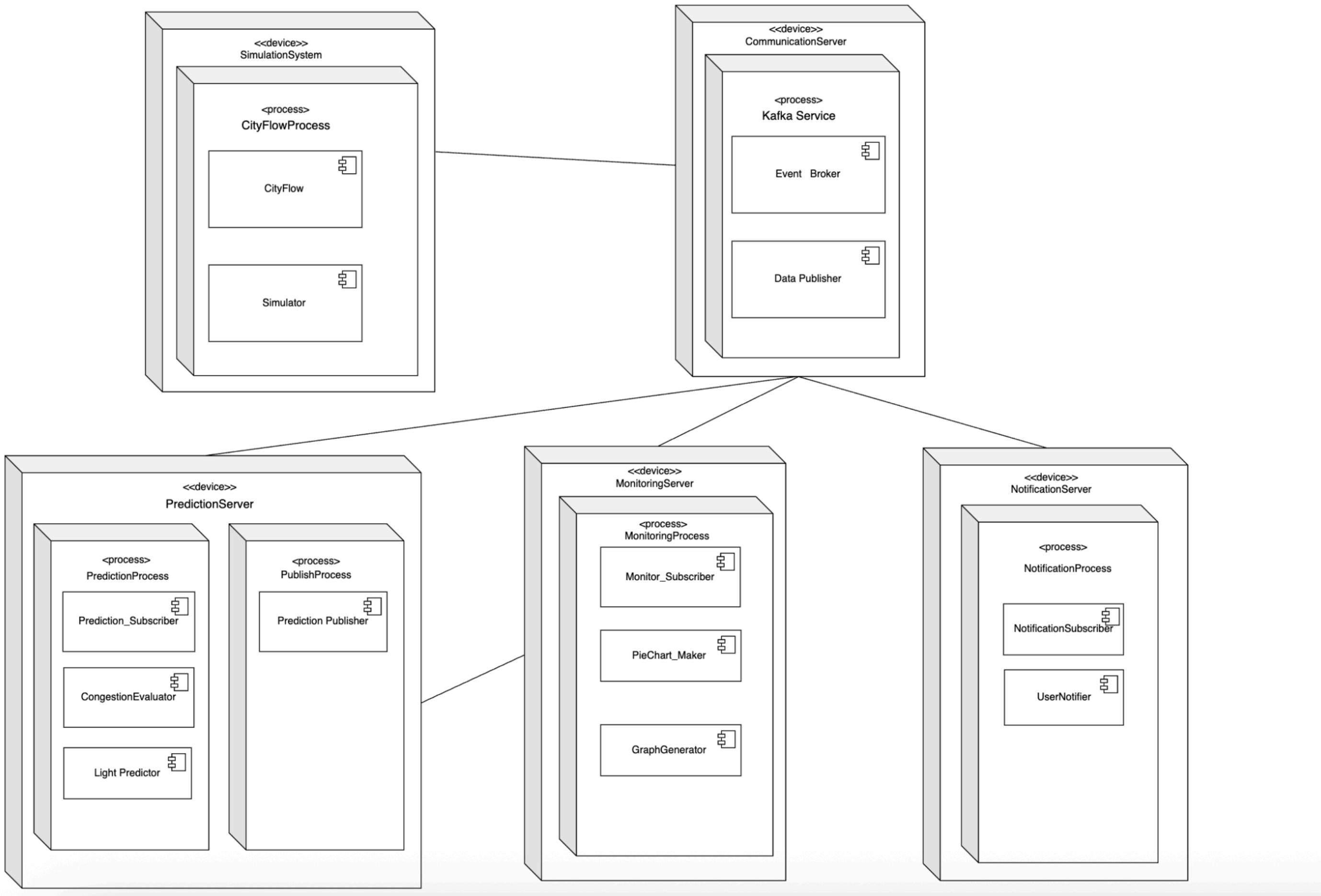



Monitoring Module
MonitoringController
- consumer: KafkaConsumerAdapter
- poller: Poller
- buffer: DataBuffer
- analytics: AnalyticsEngine
+ getMonitoring(): Any
KafkaConsumerAdapter
- topics: String[ ]
- groupId: String
+ startConsuming(): void
+ onMessage(msg: Any): void
Poller
- apiUrl: String
- intervalSec: Int
+ startPolling(): void
+ fetchPrediction(): Any
AnalyticsEngine
+ computeVehicleStats(): Any
+ computeGreenAnalytics(): Map<String,Float>
DataBuffer
- rawHistory: List<Any>
- predHistory: List<Any>
- maxSize: Int
+ addRaw(snapshot: Any): void
+ addPred(pred: Any): void

2. C&C View


CityFlow
CityFlow_Engine
Kakfka_Publisher
Simulator
Kafka_Consumer
Simulation_logic
WebSocket_Handler
Kakfka_Publisher
Prediction
Kafka_Consumer
LSTM_Light_Predictor
Kafka_Publisher
CongestionEvaluator
API
Monitoring
Kafka_Consumer
Analytics_Engine
API
Notificati...
Kafka_Consumer
Email_Sender
API
API_Gateway
Downstream_API
Frontend
UI_Components

3. Allocation View



Benefits of Microservice Architecture

- **Modularity & Scalability:** Each service can be developed, scaled, and deployed independently.
- **Fault Isolation:** Service failures are contained; the API Gateway can route around unhealthy instances.
- **Technology Heterogeneity:** Teams may choose the most appropriate stack per service (e.g., different languages or databases).
- **Maintainability & Testability:** Clear service boundaries and well-defined APIs simplify testing, versioning, and rolling upgrades.
- **Resilience & Performance:** Asynchronous messaging decouples producer/consumer lifecycles and smooths traffic spikes.

Benefits of SOAP Architecture

- **Platform & Language Neutral**
    - XML-based messaging works across any OS or language runtime. • Formal Contract (WSDL)
    - Precise service definitions enable tool-driven stub generation and early validation.
- **Extensibility**
    - SOAP headers allow adding features (security, transactions, routing) without altering payload.
- **Built-in Error Handling**
    - Standardized `<Fault>` element conveys rich error codes and diagnostics.
    - WS-Security Compliance
    - Integrates with WS-Security for message integrity, confidentiality and authentication.
- **Reliability & Transactions**
    - Works with WS-ReliableMessaging and WS-AtomicTransaction for guaranteed delivery and coordinated commits.
- **Firewall & Proxy Friendliness**
    - Uses HTTP/HTTPS as transport, easing traversal through existing infrastructure.

Microservice architecture Quality Attributes & Tactics

Priority	Quality Attribute	Tactics
1	**Availability**	• Replication (≥2 instances/service behind LB) • Health checks & auto-failover • Circuit breaker
2	**Performance**	• Async messaging via Kafka • Local caching of hot state • Bulkhead isolation (dedicated threads)
3	**Scalability**	• Stateless Docker services for horizontal scale • Partitioned Kafka topics • Auto-scaling groups • Docker orchestration
4	**Security**	• E2E encryption (Fernet/Base64) on Kafka • API-Gateway AuthN/Z (JWT/RBAC) • TLS everywhere
5	**Modifiability**	• Strong contracts (WSDL, OpenAPI) • Semantic versioning of APIs • CI/CD with rolling upgrades

SOAP architecture Quality Attributes & Tactics

Priority	Quality Attribute	Tactics
1	**Interoperability**	• WSDL-first contract to auto-generate language-neutral stubs • XML Schema validation for payload consistency • SOAP headers for extensible metadata
2	**Security**	• WS-Security (XML-Signature, XML-Encrypt) on SOAP envelopes• TLS at transport layer • UsernameToken or X.509 token profiles
3	**Reliability**	• WS-ReliableMessaging for guaranteed delivery and ordering • Idempotent operations (replay safe) • Retry policies with back-off on faults
4	**Performance**	• MTOM to optimize binary payloads • HTTP Keep-Alive & connection pooling • Message compression (gzip on XML)
5	**Modifiability**	• Extensible headers for non-functional metadata • Loose coupling via versioned WSDL namespaces • Schema evolution with optional elements