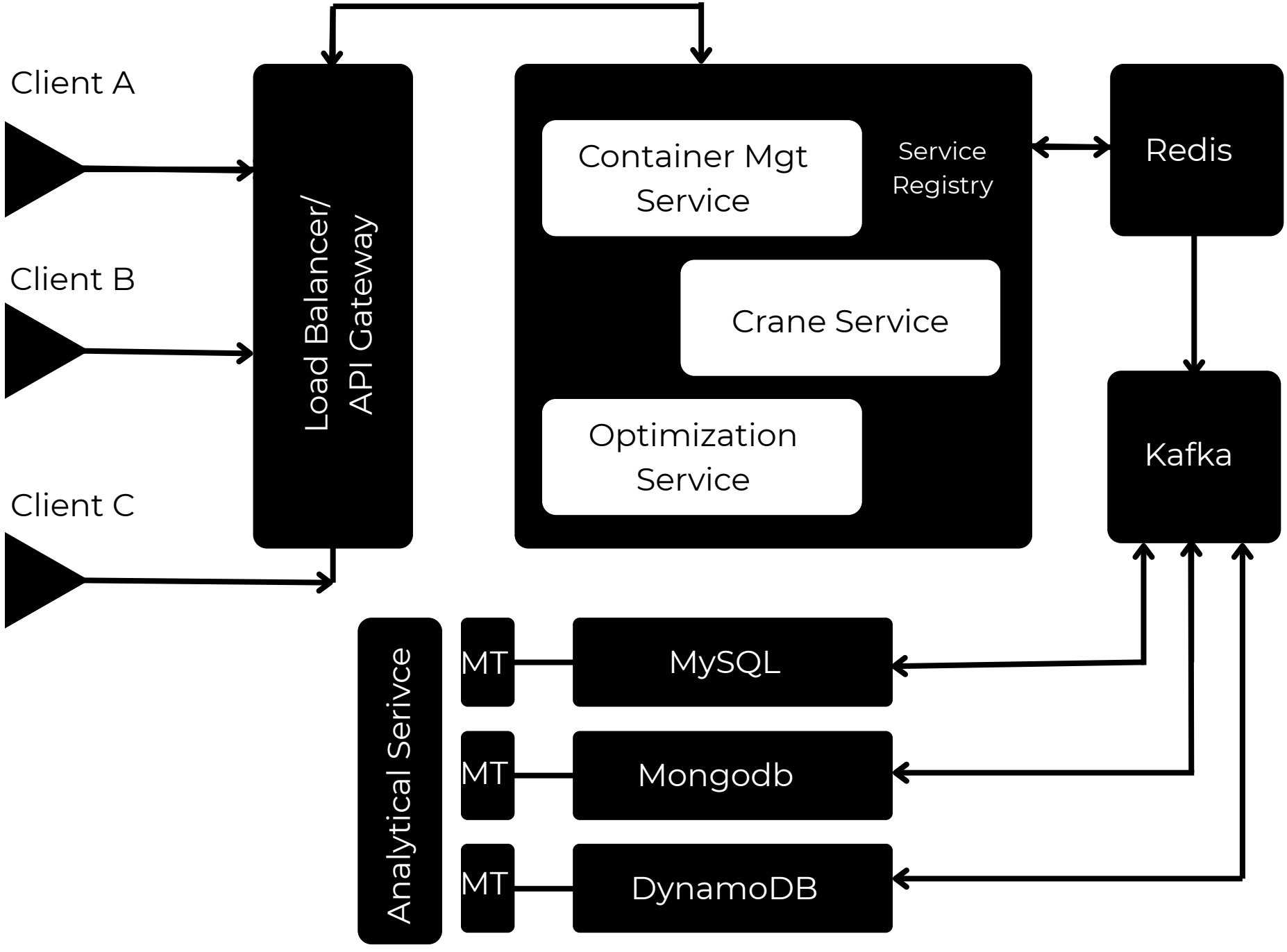
Client A
Client B
Client C
Load Balancer/
API Gateway
Container Mgt
Service
Service
Registry
Crane Service
Optimization
Service
Redis
Kafka
Analytical Serivce
MT
MySQL
MT
Mongodb
MT
DynamoDB

Introduction

In this technical assignment, I designed a backend system for managing crane operations, container movement, and yard optimization. The system focuses on real-time data processing, scalable service communication, and efficient resource usage.

Tech Stack

- **Load Balancer/API Gateway** -Distributes incoming traffic across multiple instances of API Gateway Entry point for Admin Panels, Sensors
- **Eureka Serve**r -Enables services to register and discover each other dynamically
- **Service Registry(Eureka client and server)** - Container Management, Crane Service, Optimization Service having Core business logic
- **Kafka** -Message broker for asynchronous event flow
- **Redis** -Caches frequently accessed data
- **Multiple Databases(Dynamodb, Mysql, MongoDb)** - Separate DBs for Container, Crane, Yard, etc.
- **Metadata Trackers(MT'S)**-Attached to each database to track record-level metadata like createdBy, lastUpdatedAt
- **Analytics Service-** Consumes data from all DBs + metadata to produce insights

System Architecture Diagram

Component Descriptions

API Gateway

- Single entry point for all incoming requests (Admin Panel, Sensors).

Container Management Service

- Handles container tracking and management tasks.

Crane Service

- Communicates crane positions, health, and status updates.

Optimization Service

- Suggests best yard allocations and crane scheduling.

Kafka

- Event bus that streams messages between services asynchronously.

Redis Cache

- Caches recently accessed data, reducing read times.

Databases

- Persist detailed container, crane, yard, and event logs separately.

Analytics Service

- Reads from databases to produce operational dashboards and predictions.

Redis Integration

Redis is integrated between microservices and the database layer. Whenever services need to retrieve frequent operational data, they first query Redis. If data is not available, they fallback to database queries. This reduces database load and improves overall latency.

Final Reflection

Through this assignment, I learned about designing scalable backend systems using event-driven architecture. I understood the importance of asynchronous communication (Kafka), caching mechanisms (Redis), real-time analytics, and database optimizations to handle high-throughput industrial environments.