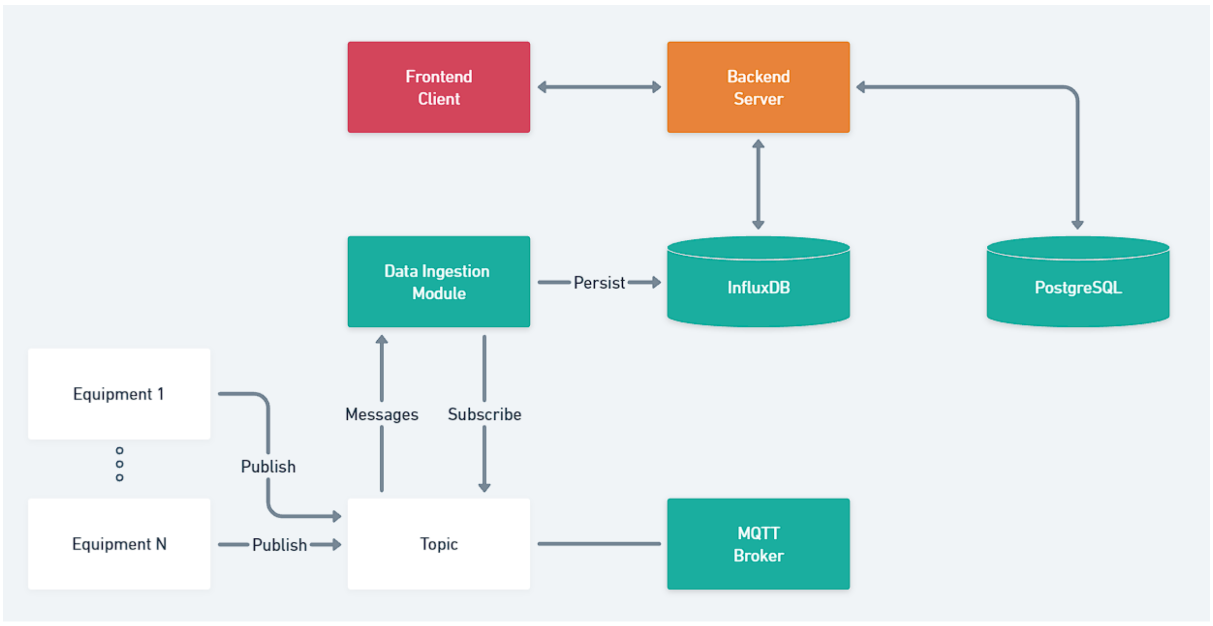
Radix Challenge

Intro

The diagram below shows a system designed for data collection, processing, and storage, utilizing various technologies and components to facilitate these processes. It highlights the flow of data from multiple machines through an MQTT broker, to a data ingestion module, and finally into both InfluxDB (a time-series database) and PostgreSQL (a relational database).

Architectural diagram



Architectural description

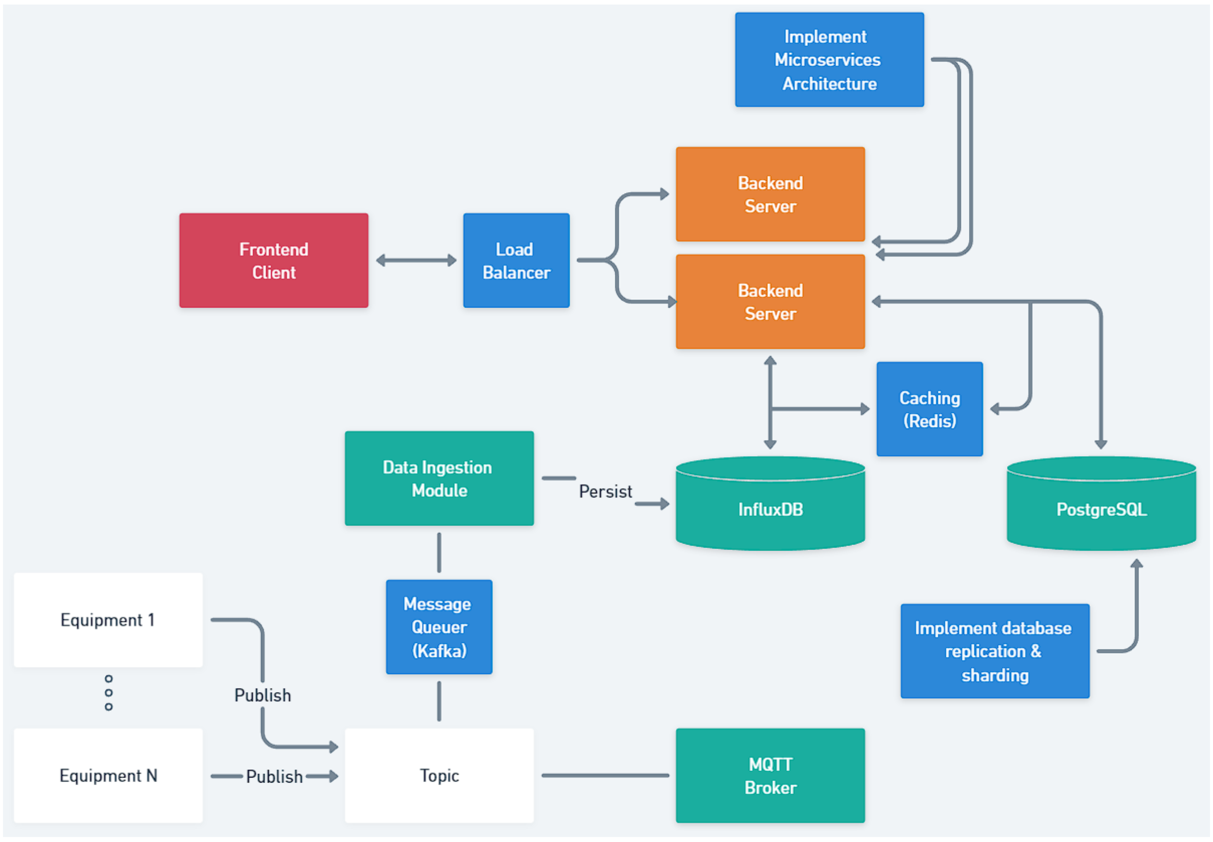
Components & Interactions

1. **Frontend Client**: web application, serving as the user interface for interacting with the system.
2. **Backend Server**: Handles requests from the frontend client. It provides APIs for data retrieval and management. This could eventually become a

"control room" for the available equipments.

3. **Equipment 1 … Equipment N:** These symbolize various devices or machines that generate data.
4. **MQTT Broker:** A central message broker that employs the MQTT protocol for lightweight publish-subscribe messaging. It receives data published by the machines and facilitates its delivery to subscribers.
5. **Topic:** A named channel on the MQTT broker where messages are published and subscribed to. Machines publish data to a specific topic, and the data ingestion module subscribes to it.
6. **Data Ingestion Module:** The data ingestion module is responsible for persisting (storing) the processed data into the InfluxDB time-series database. This component subscribes to the MQTT topic, receives data from the machines, and processes it. This could eventually become complex due to possible necessities involving data cleaning, transformation, and preparation for storage.
7. **InfluxDB**: stores time-stamped data, often used for metrics, sensor data, and monitoring.
8. **PostgreSQL:** A traditional relational database, suitable for storing structured data with relationships between entities. It manages our users and equipments. Could possibly be responsible for controlling configuration over machines in the future, despite additional managing and monitoring features.

Suggestions for improvemenets over architecture



- **Load Balancer**: Like a traffic cop, it directs incoming requests to different servers, so no single one gets overloaded.
- **Message Queue**: A waiting room for data. It helps when your equipment sends data faster than your system can handle it, preventing data loss.
- **Microservices**: Instead of one big backend server, break it into smaller pieces, each doing one specific job. This way, you can scale up only the parts that need it.
- **Caching**: Keep frequently used data in a super-fast storage (like a cache), so you don't have to keep asking the database, making things snappier.
- **Database Tricks** (replication and sharding): Make copies of your databases or split them up if they get too big. This helps them stay fast even with tons of data.