Team 5 - Data Collection, Processing, and Visualization for Remote Emergency Generator Systems

UCONN SCHOOL OF ENGINEERING
COMPUTER SCIENCE & ENGINEERING

Team Members: Evan Langlais, Emil Abraham, William Reid, Rania Chowdhury
Sponsor Advisor: William Schneeloch
Sponsored by: Kinsley Power Systems
Faculty Advisor: Dr. Song Han

Project Sponsor

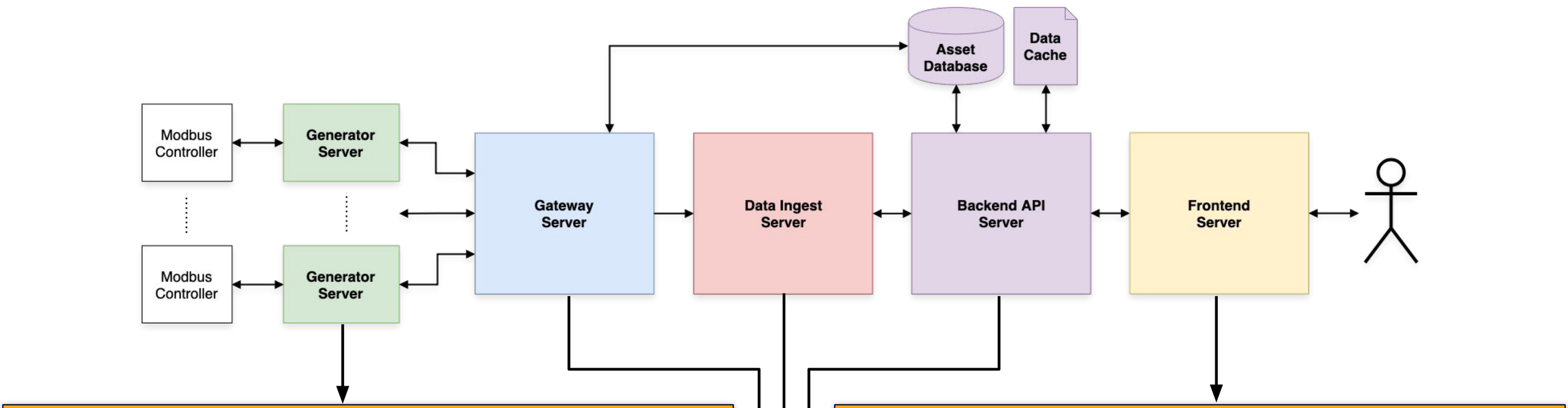
KINSLEY
The Energy Solutions Company

Project Motivation

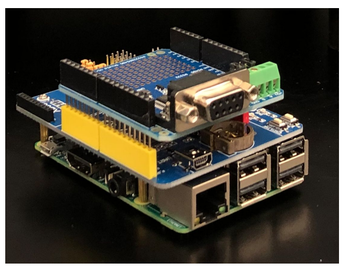
Generators are a cornerstone of emergency power infrastructure that ensure critical systems are up and running during power failures. Currently, a technician must physically travel to each individual generator to view its health and current sensor data, and cannot deduce trends and macroscopic issues as each generator is isolated from each other. Our project sets out to rectify both of these issues by enabling remote monitoring of generator health and sensor data, and implementing a centralized database of sensor data / fault history for all generators to be analyzed and visualized.

System Overview

After discussion with the sponsor, it became clear that our system would be similar to other Big Data / Internet of Things (IOT) system solutions. The source of data originates from many independent and distributed servers, in our case connected to a generator controllers, polling and then pushing data into a central ingest server to be further processed and used in meaningful services. To ensure an agnostic environment for future production deployment to a cloud provider, our system's datapath was designed to run on Red Hat Enterprise Linux (RHEL) servers. Each component of the datapath has a particular role in the transformation of raw data into meaningful data analytics and visualization via the website frontend. Building a system to handle thousands of concurrent generator connections with gigabytes worth of data being ingested every day is ultimately what lead to the following system design.



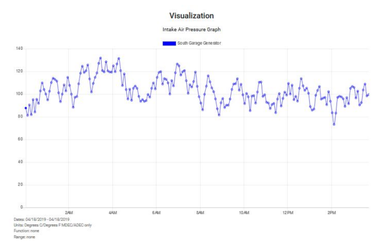
Generator Server

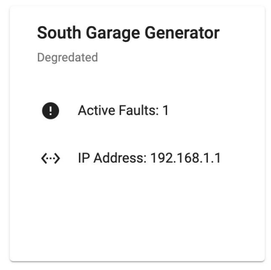


The Generator Server is running on a Raspberry Pi connected to the Controller's Modbus RS-485 port via a USB to serial converter.

The server is written in Go. It establishes a WebSocket connection to the Gateway Server for bi-directional communication. It constantly reads data and status registers, aggregates them, and periodically sends them to the gateway server, which responds with a verification packet. If none is received, the data is re-sent with the next packet.

Frontend Server



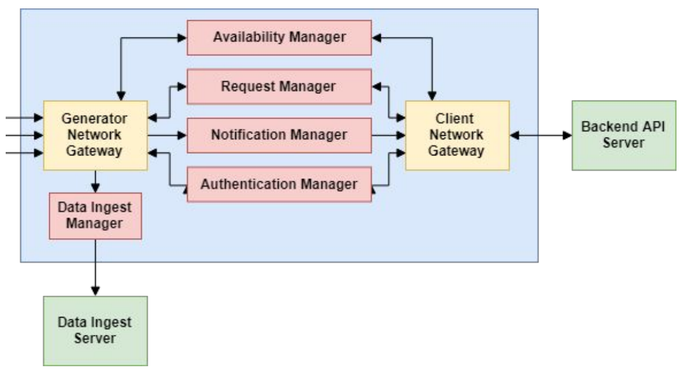


The visualization page plots a specified data metric for up to five generators over a chosen time period with optional downsampling.

The statistics page serves to display the most recent generator data metrics and generator faults and statuses.

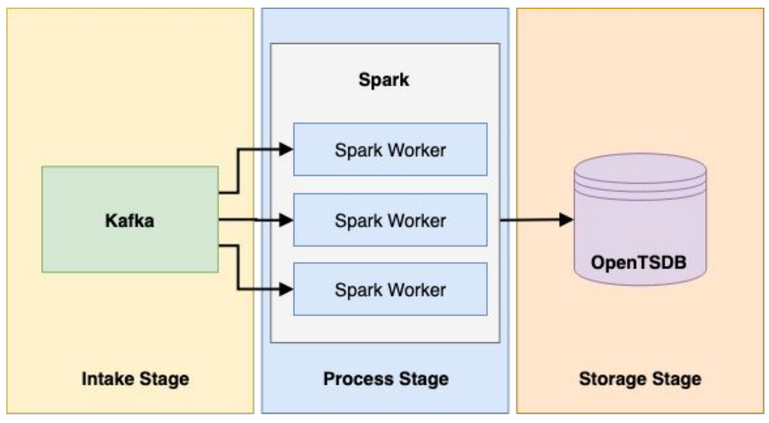
The dashboard page shows which generators a user has access to and a quick overview of that generator's health and information.

Gateway Server



The Gateway Server is written in Go, and accepts incoming WebSocket connections over TLS. It maintains each generator's connection, metadata, and state in a centralized MySQL database. Data packets received by the Gateway are serialized with Avro and sent to Kafka for processing. When status code update packets are received, the required state change is updated in the MySQL database and any necessary notifications are sent to technicians.

Data Ingest Server



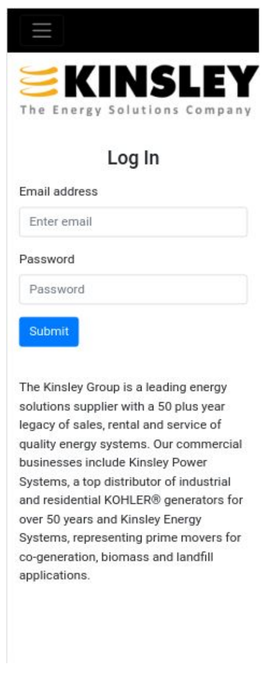
Intake - Data enters via Kafka and is transformed into meaningful data using Avro

Process - Data processed via Spark running an application written in Scala

Storage - Processed data pushed into OpenTSDB via a WebSocket for storage and future analysis.

Backend API Server

Data is queried by Express, a Node.js web application. Express queries data from OpenTSDB for generator metric data and MySQL for user and generator information. Through these queries, the frontend server can call endpoints in order to appropriately display the relevant data on the website.



The server is also responsible for establishing an authentication system for the website. This allows for authorized viewing of generator data.