**Individual Project 2: Streaming Data Pipeline with Pub/Sub Architecture**

**CS367**

**Cassandra Mason**

**10/24/24**

Table of Contents

[Project Description 3](#_Toc180665885)

[Pub/Sub Architecture 5](#_Toc180665886)

[Pseudocode of Pub/Sub Architecture 5](#_Toc180665887)

[Coding Implementation 8](#_Toc180665888)

[Results 10](#_Toc180665889)

[Conclusion 11](#_Toc180665890)

[References 11](#_Toc180665891)

# Project Description

This is a design proposal and simple implementation of a streaming data platform for a renewable energy company, REC. This is a fictitious company used to demonstrate the concept and utility of streaming data. The design focuses on publish subscribe (pub/sub) architecture to stream energy production data, weather data, equipment data, and grid data for the company to leverage in decision-making. It highlights the sources of data (producers), the streaming data pipeline, and the places where the data is analyzed and used (subscribers). It will include a description of the system, a visual of the pipeline, the pseudocode design of the pub/sub architecture, the coding implementation of the pub/sub architecture, the results of such a design, and a conclusion.

REC is a renewable energy company that has wind and solar farms across the United States. The wind turbines and solar panels are constantly sending data about their energy output and equipment functions. REC also has small weather monitoring stations at every site that constantly measures temperature, humidity, wind speed, wind direction and other weather conditions that may impact the equipment and/or energy production.

The company needs data from these sources to respond to events in real time and make business decisions about their company. They need to know how much energy they are producing, from which sources, how much energy the grid is demanding, the effect of weather on energy production/consumption, when there are equipment malfunctions, how to maximize efficiency, how to grow their business, etc. As well as the data their proprietary equipment produces, REC would also like to pull in data from weather stations and the grid at-large to help answer some of these questions.

What is pub/sub architecture? Focus on stream vs static data.

The data sources in this project are the publishers in the pub/sub streaming data architecture. For REC, the data sources are:

* REC wind turbines
* REC solar panels
* REC weather stations
* National weather data streaming database
* National grid data streaming database

The turbines and solar panels produce power output data as well as monitoring data about the status and performance of the equipment to inform maintenance requirements. The weather sensors provide data on the local conditions that may effect the equipment. The weather database gives a broader picture of weather conditions and their impact on both energy production and energy consumption (when paired with the energy data). The grid data reveals information about production and consumption of electricity.

The REC devices are Industrial Internet of Things (IIoT). Their data is packaged into JSON files at the device, then sent over a network, such as cellular, ethernet, or wifi. It is transferred using a standardized communication protocol such as HTTPS. The data is received by a message broker, such as Apache Kafka, to hold the publisher data for the subscribing applications.

To access the source databases requires an external API that ingests their data into this pipeline. Regular API calls to their databases allow for a near continuous stream of data to match the continuous stream of data coming in from the REC devices.

Data is divided into topics, and subscribers subscribe to specific topics of interest. The topics for this system are:

* Energy-production
* Weather-data
* Equipment-status
* Grid-status

The energy-production topic contains data about power output for solar panels and wind turbines. Overall energy consumption, energy production, the mix of different energy sources, and carbon dioxide emissions.

The data users in this project are the subscribers in the pub/sub streaming data architecture. They subscribe to certain topics and ingest that data to provide insights and store data. The subscribers are:

* Real-time data analytics
* Streaming data dashboard
* Database for storage and historical analysis
* Alerting system

Real time analytics process the data as it arrives to gain insights. Apache Spark is a tool to use to analyze streaming data. REC wants to use real time analytics with machine learning to predict how/when equipment failures or shutdowns may occur and predict the effect of weather on energy production and consumption. Streaming data dashboards will visually show data as it arrives: power production, grid load, rolling averages, temperature, cloud cover, etc. This will help REC and their customers understand their energy landscape. A storage database will subscribe to the message broker to store historical data for historical analysis, reporting, and trend tracking. Such a database could be Amazon S3 or Hadoop Distributed File System. The last subscriber is a system that monitors the equipment status and weather to trigger real-time alerts for maintenance teams when equipment malfunctions, fails, or other anomalies in the data.

# Pub/Sub Architecture

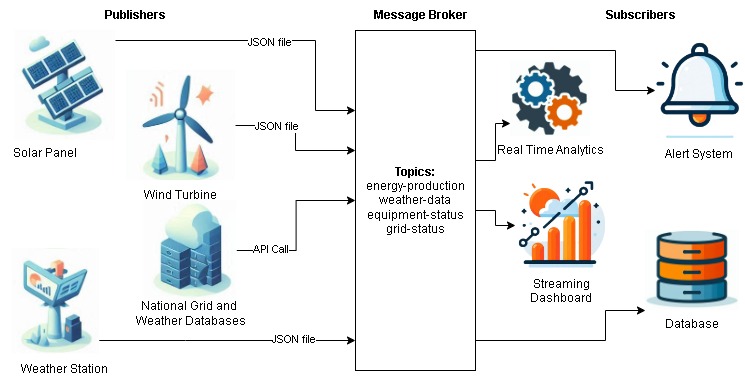


Image generated with draw.io (n.d.).

This flowchart shows the pub/sub architecture of the REC streaming software design. It includes the publishers, message broker with message topics, and the subscribers. The publishers are the solar panels, wind turbines, weather stations, and databases for national weather and grid data. The IIoT devices send information in JSON files, while the database data is accessed with regular API calls. The topics are enery-production, weather-data, equipment-status and grid-status. The subscribers to these topics are real time analytics applications, an alert system, streaming dashboards, and a database to store data.

# Pseudocode of Pub/Sub Architecture

Pseudocode to create a publisher class:

CLASS Publisher:

FUNCTION initialize(broker, topic):

SET broker to broker

SET topic to topic

FUNCTION generate\_data():

WHILE True:

data = collect\_data()

publish\_to\_broker(topic, data)

WAIT for 1 second

FUNCTION publish\_to\_broker(topic, data):

SEND data to broker under topic

PRINT “Data published to topic: “, topic

END CLASS

CLASS Subscriber:

FUNCTION initialize (broker, topic):

SET broker to broker

SET topic to topic

FUNCTION subscribe(topic):

WHILE True:

messages = pull\_data\_from\_broker(topic)

FOR each message IN messages:

process\_message(message)

WAIT for 1 second

FUNCTION pull\_data\_from\_broker(topic):

RETURN GET messages from broker under topic

FUNCTION process\_message(message):

PRINT “Processing message: “, message

END CLASS

This pseudocode sets up the publisher and subscriber classes. The publisher class contains several functions. The “initialize” function creates an instance of a publisher that sets the broker it connects with and the topic(s) it publishes to. The “generate\_data” function pulls data from the device or database, assigns it to a topic, and sends it to the message broker using the function “publish\_to\_broker”. The subscriber class contains functions to initialize an instance of a subscriber and subscribe to certain topics. The subscribe function calls a function that pulls the messages from a certain topic from the broker and processes the messages.

Pseudocode to initialize the publishers in the REC streaming software system and begin the streaming process:

Solar\_panel = Publisher(broker = “message\_broker\_url”, topic = [“energy-production”, “equipment-status”])

CALL Solar\_panel.generate\_data()

Wind\_turbine = Publisher(broker = “message\_broker\_url”, topic = [“energy-production”, “equipment-status”])

CALL Wind\_turbine.generate\_data()

Weather\_station = Publisher(broker = “message\_broker\_url”, topic = [“weather-data”, “equipment-status”])

CALL Weather\_station.generate\_data()

Grid\_database = Publisher(broker = “message\_broker\_url”, topic = [“energy-production”, “grid-status”])

CALL Grid\_database.generate\_data()

weather\_database = Publisher(broker = “message\_broker\_url”, topic = “weather\_data”)

CALL weather\_database.generate\_data()

Each publisher is denoted by name and assigned to the relevant topics for the type of data it produces.

Below is the pseudocode for initializing the subscribers in the system and starting the consuming process.

alert\_system = Subscriber(broker = “message\_broker\_url”, topic = [“equipment-status”, “weather-data”]

CALL alert\_system.subscribe(“equipment-status”)

CALL alert\_system.subscribe(“weather-data”)

streaming\_dashboard = Subscriber(broker = “message\_broker\_url”, topic = [“equipment-status”, “weather-data”, “energy-production”, “grid-status”]

CALL streaming\_dashboard.subscribe(“equipment-status”)

CALL streaming\_dashboard.subscribe(“weather-data”)

CALL streaming\_dashboard.subscribe(“energy-production”)

CALL streaming\_dashboard.subscribe(“grid-status”)

analytics = Subscriber(broker = “message\_broker\_url”, topic = [“equipment-status”, “weather-data”, “energy-production”, “grid-status”]

CALL analytics.subscribe(“equipment-status”)

CALL analytics.subscribe(“weather-data”)

CALL analytics.subscribe(“energy-production”)

CALL analytics.subscribe(“grid-status”)

database\_storage = Subscriber(broker = “message\_broker\_url”, topic = [“equipment-status”, “weather-data”, “energy-production”, “grid-status”]

CALL database\_storage.subscribe(“equipment-status”)

CALL database\_storage.subscribe(“weather-data”)

CALL database\_storage.subscribe(“energy-production”)

CALL database\_storage.subscribe(“grid-status”)

The above pseudocode initializes all the connections to subscribing applications referencing the message broker and the topics they subscribe to. It then starts the flow of data into these applications by calling the subscribe function for each topic the subscriber follows.

# Coding Implementation

Install Kafka and start zookeeper in a command prompt window:

A screenshot of a computer

Description automatically generated

Start the Kafka Broker in a command prompt window:



Create the Kafka topics for the REC Kafka message broker in a command prompt window:

A screenshot of a computer program

Description automatically generated

Once Kafka is up and running, in Spyder I imported the libraries needed to communicate with kafka and created a variable storing the port number where kafka is running:

A screenshot of a computer

Description automatically generated

Create a class for producers:

A screen shot of a computer program

Description automatically generated

Create a class for subscribers:

A screen shot of a computer code

Description automatically generated

Instantiate the different producers as their own objects:

A screen shot of a computer code

Description automatically generated

Start an example data stream from the solar panel object:



(*Apache Kafka*, n.d.; *Kafka-Python*, n.d.; Contributor, 2024; Simplilearn, 2017)

# Results

This project was implemented using Kafka as the message broker between the publishers and subscribers. A publish and subscribe class were created to make the relevant instances of publishers and subscribers to the REC system.

I ran out of time to make the code work, I need to define a variable ‘topic’ as well as topics, so when I ran the code I have, I got this error:

A screen shot of a computer program

Description automatically generated

Currently a simple data stream model is used to replicate data. Future work would include using API’s for streaming databases to pull in real-world, relevant data streams. The API’s of interest are OpenWeather’s Weather API (*Weather API - OpenWeatherMap*, n.d.), NREL’s renewable energy database (*NREL: Developer Network*, n.d.), and Electricity Maps for grid data (*Get Our Data | Electricity Maps*, n.d.).

Future work also includes creating a scatterplot visual of incoming energy production, temperature, and energy consumption data using Apache Spark. This would show a rolling window of streaming data starting from the most recent data to a day back. It would be part of the streaming dashboard subscriber application, and benefit users by showing them in real time what data is being pulled into the system, and what values they have.

I also want to run the different producers on separate threads so they can all run simultaneously.

# Conclusion

This pub/sub streaming data software design for the renewable energy company, REC, allows for real-time data to inform decisions about energy production, grid utilization, and the effect of weather on supply and demand. The publishers that provide streaming data include renewable energy devices: solar panels and wind turbines, on-site weather stations, and streaming databases about overall weather and grid usage.

They send data to different topics. The energy production topic can tell us how much power is generated by solar panels, wind turbines, the total power output to the grid, etc. The equipment status topic lets maintenance teams at REC monitor the equipment for failures, malfunctions, wearing, and unsafe weather conditions for equipment operation that lets them take care of the equipment better. The weather data topic allows for prediction models to determine the effect of weather conditions on power production and consumption, and what weather variables are unsafe for equipment operation. The grid-status topic gives information about overall supply and demand, when and where power is most needed, where there are surpluses, what the overall makeup of power generation is (renewable vs. non renewable) and long-term strategies for grid optimization.

Subscribers to these topics process the data streams to create visual dashboards about what is happening with REC’s equipment and power generation, grid production and consumption, and weather trends. An alerting system subscriber monitors the equipment status data and automatically sends alerts or shuts down devices when there is a red flag. Real-time analytics looks for trends and makes predictions from the incoming data streams. The processed data is stored in a database for historical analysis, record keeping, and reports.

# References

*Apache Kafka*. (n.d.). Apache Kafka. Retrieved October 24, 2024, from https://kafka.apache.org/downloads

Contributor, G. (2024, January 29). *How to install and configure Apache Kafka on Windows*. Sling Academy. https://www.slingacademy.com/article/how-to-install-and-configure-apache-kafka-on-windows/

*Draw.io*. (n.d.). Retrieved August 23, 2024, from https://www.drawio.com/

*Get our data | Electricity Maps*. (n.d.). Retrieved October 24, 2024, from https://www.electricitymaps.com/get-our-data

*kafka-python: Pure Python client for Apache Kafka* (Version 2.0.2). (n.d.). [Python]. Retrieved October 24, 2024, from https://github.com/dpkp/kafka-python

*NREL: Developer Network*. (n.d.). Retrieved October 24, 2024, from https://developer.nrel.gov/

Simplilearn (Director). (2017, July 25). *Kafka Tutorial | Apache Kafka Tutorial For Beginners | Kafka Architecture |What Is Kafka|Simplilearn* [Video recording]. https://www.youtube.com/watch?v=U4y2R3v9tlY

*Weather API - OpenWeatherMap*. (n.d.). Retrieved October 24, 2024, from https://openweathermap.org/api