SpecPipe:
A scalable cloud-based AI/ML-facilitating data pipeline for spectrum

Zhen Jiang, Victor Li, Shruti Satrawada, Sivani Voruganti, Gen Yang

May 2023

1 Executive Summary

This paper proposes an adaptable end-to-end data pipeline for radio spectrum data that allows interested practitioners or the general public to harness, understand, and utilize distributed radio spectrum data without access to company-level resources. Currently, there is no mature end-to-end solution for access to distributed spectrum data, which is complex and dynamic in nature, with additional requirements for custom post-processing depending on the intended use.

Spectrum is a resource consisting of the range of electromagnetic radiation used to transmit information wirelessly. The radio frequency spectrum powers all communication around us–from cell phones to WiFi and more. There are already a myriad of streams of this data being broadcasted without people being aware that this information is accessible and can be used for new and innovative applications. Even if people are aware of this data, reading, processing, and utilizing it is not straightforward–which is exactly what our project is meant to address with our customizable pipeline and documentation.

The pipeline is designed to transform raw human-unreadable radio spectrum data into a final user-facing application or visualization, allowing users to process diverse types of spectrum data while still maintaining necessary security and control over private data that they may feed into their pipelines.The proposed solution pipeline is lightweight, portable, and robust, enabled by utilizing affordable state-of-the-art cloud technologies for risk and fault tolerance.

We also provide detailed documentation and start-from-scratch tutorials, enabling audiences of diverse experience levels to build their own custom spectrum pipelines, such as radio content analyzers and signal monitors.

The paper describes the design choices by building out an example Airplane Tracker application built on top of the pipeline from beginning to end, showcasing how the pipeline can be adapted to process diverse data types and improve accessibility to spectrum resources.

The paper provides valuable insights into designing distributed spectrum data pipelines and their potential for increasing user accessibility to spectrum resources. Our entire work is open-source, customizable, and affordable, making it accessible to motivated individuals without the need for extensive resources. The proposed directions for future work include additional user studies and testing to validate the effectiveness of the tutorials and further refinement of the data pipeline to enhance its efficiency and accuracy. Overall, this work takes steps towards facilitating the development of more effective and user-friendly applications in various domains such as telecommunications and security, ultimately democratizing access to spectrum data and enabling a broader range of stakeholders to leverage this valuable resource for their own purposes.

2 Introduction

2.1 Problem & Opportunities

Radio Spectrum is a resource consisting of the range of electromagnetic radiation frequencies used to transmit information wirelessly. The radio frequency spectrum powers much of the communication around us—from FM, to aircraft safety and communications, to WiFi and 5G signals for mobile phones. With important information communicated every millisecond, spectrum data is essential and there are limitless applications that can be built using spectrum data.

Currently, there is no mature and affordable end-to-end solution for the general public to harness, understand, and utilize distributed radio spectrum data without access to company-level resources. This is due to the fact that Wireless Spectrum data is complex and dynamic in nature. A custom pipeline is required to handle the massive volume and diversity of the data, while still ensuring high quality and reliability. Privacy is another impor-

tant consideration when it comes to spectrum data, as the data often contains private or locational information that requires careful treatment that cannot be found in un-customizable, public pipeline solutions. Finally, it is important to note that there are already several levels of solutions at various price points for spectrum monitoring for a single location. However, if a user wants access to distributed spectrum data that is being read from sensors scattered across several different geographical regions–there are not as many solutions available. This key pain point, when combined with the aforementioned requirements of cost, privacy, usability, high quality, and reliability, necessitates a new system to ensure these qualities, and we introduce our project, SpecPipe.

2.2 Our Proposed Solution

Our project, SpecPipe, proposes 1) an adaptable end-to-end data pipeline for radio spectrum data, 2) detailed tutorials for audiences of diverse experience levels to build their own custom spectrum pipelines, and 3) An example Airplane Tracker application built upon our pipeline to showcase how our pipeline can be adapted to process diverse data types, to ultimately improve public accessibility to spectrum data.

Our primary product is the end-to-end spectrum data pipeline or automated processing solution, which consists of several modularized components working together to transform raw human-unreadable radio spectrum data into a final user-facing application or visualization. In order to develop this pipeline, we utilize affordable state-of-the-art cloud technologies for risk and fault tolerance as well as scaling, in order to make our pipeline more robust. Risk tolerance involves assessing and managing potential threats to the functionality of the pipeline, while fault tolerance ensures that the system can continue functioning despite failures or at the very least fail gracefully. Scaling is especially important for our use case since our pipeline should be able to support data at a kb/s rate from tens of devices cheaply and be able to scale to support higher data rates from thousands of sensors. Our data pipeline is ultimately lightweight, portable, robust, observable, and intelligent–permitting easy integration with current AI/ML analytical engines.

Alongside our technical contributions, we maintain a unique focus on prioritizing user-friendliness and the accessibility of our technology to broader audiences by providing detailed documentation and start-from-scratch tutorials. As our long-term vision, we aim to make traditionally under-utilized spectrum data more accessible to our target audience of researchers and hobbyists, as well as the general public at large, for their own applications and purposes.

Alongside our data pipeline product and documentation/tutorials, we also introduce an Airplane Tracker as an example of a domain-specific spectrum application that users can build on top of our pipeline. The Airplane Tracker works with airplanes in flight emitting a certain form of radio data that occupies a specific portion of the spectrum. This example of a domain-specific spectrum application showcases how our data pipeline can be adapted to process this data and ultimately power an online Airplane Tracker application that allows you to see where planes are flying on an interactive map.

2.3 Value Claims

Our end-to-end data pipeline is the first of its kind in being open-source, customizable, and affordable, and is designed especially for motivated individuals to be able to replicate, adapt, and utilize it–without the need for extensive company-level resources. Ultimately, the flexibility of our pipeline allows users to process diverse types of spectrum data, while still maintaining necessary security and control over private data that they may feed into their pipelines.

Key Properties	Description
Affordability	RTL-SDR & AWS resources are less expensive than alternatives.
Lightweight	• *Pipeline Modules:* Dockerhub and Kubernetes allow consistent, lightweight, and portable deployment across different computing environments. • *Pipeline Messaging System:* NATS-based data distribution backbone provides a light pipeline. NATS is considered lightweight due to its small footprint, low overhead, and efficient communication protocol, making it easy to integrate into existing systems and ideal for use in distributed systems and microservices architectures.
Robustness Reliability Scalability	A custom pipeline can handle the massive volume and diversity of the data, while still ensuring high quality.
Privacy	By making a product meant to be re-created and customized, users can handle their own data and do not need to trust their data with any external organization.
Observability	Our System Dashboard provides visibility into various components of the pipeline and their status.
Replicability Adaptability Accessibility User-Friendly Open-Source	Our Documentation, Tutorials, and Airplane Tracker Example Application help motivated individuals replicate our pipeline and adapt it towards their own purposes.

Figure 1: Key Properties and how they are ensured in SpecPipe

3 Literature Review

3.1 Industry Connections

Our SpecPipe project has connections with the communications and PaaS (Platform-as-a-service) industries. The communications industry, or the telecom industry, provides network and computing services. The PaaS industry tries to build "a complete development and deployment environment in the cloud, with resources that enable organizations to deliver everything from simple cloud-based apps to sophisticated, cloud-enabled enterprise applications" (Gómez). These industries are deeply intertwined with our project since we are building a general multi-functional data pipeline in the cloud that is easy to access and use. Our cloud-based design enables access to core pipeline services from anywhere in the world, and also makes the pipeline easily scalable.

3.2 Relevant Approaches in Data Pipeline Design

We first examine modern data pipeline implementations. One example is Apache Kafka which is a "publish subscribe messaging system... that is scalable, durable, and fault tolerant" (Sharvari and Sowmya 2019). While Kafka is used by many it is not ideal as it cannot be used in cases when real-time data is critical and messages cannot be lost (Sharvari and Sowmya 2019, Bhat and Bhat 2020). Also, despite its strong functionality, building and fine-tuning the Kafka cluster can be a challenging process for new users and it is not lightweight, so there is a need for better solutions.

Another example is the ELK stack, consisting of Beats, Logstash, and Elasticsearch, which is a widely used data processing system. However, it only works with time series data and its complexity prevents non-professional users from using the system (Bavaskar et al. 2019).

The combination of InfluxDB and Grafana is yet another popular choice of technologies for data pipelines. However, a drawback of this approach is that InfluxDB stores data in memory and on disk, which can lead to high storage costs for large datasets. Also, InfluxDB is mainly designed for time-series data, while the spectrum-related data contains more than that.

Another important reason behind searching for alternative pipeline designs is that the diversity of representations and downstream dataproducts for spectrum data requires the ability to plug in custom processing blocks, which none of the above methods are designed to enable easily. Having covered the current approaches with data pipeline design, we will next examine the current approaches for dealing with spectrum-specific data.

3.3 Relevant Approaches in Spectrum Sensing

Aside from the design of our data pipeline, our project aims to explore ways to make the process of detecting, harnessing, and utilizing spectrum data more accessible to users. When it comes to using spectrum data, the traditional approaches fall short through being too expensive, complex, difficult to scale, hard to maintain, and lacking data accuracy, or some combination of these factors.

Recent research work has extensively examined designing and building sensors that are low cost, smaller in size, and more energy efficient, in order to reduce expenses in deploying dense spectrum sensor networks. Kleber et al. (2016) propose RadioHound, a spectrum sensing system consisting of several "client sensors, a centralized controller, and a user interface." Notably, the RadioHound system introduces a cost-efficient spectrum sensor design–an RTL-SDR (Realtek Software-Defined Radio) device hosted by a credit-card sized Raspberry Pi microcomputer. Saber et al. (2018) also employ this low-cost RTL-SDR/Raspberry Pi spectrum sensing approach and use MATLAB-Simulink software to analyze the data. The sensing methods proposed by both research works resonate with our project's key values of making both our software and hardware solutions accessible to the public, and so we adopted a similar strategy of constructing simplistic spectrum sensors with an RTL-SDR device that can be hosted by either a Raspberry Pi or the user's personal computer.

Another particularly impactful line of research in the area of spectrum sensing has been the creation of collaborative sensing networks. This idea of leveraging the collaboration of many users and sensors in order to form a larger sensing network capable of handling larger-scale sensing tasks has been well-explored in the broader domain of Internet of Things (IoT) (He et al. 2022). Within the realm of spectrum sensing, researchers have introduced novel ways to optimize data collection via these collaborative networks. Smith et al. (2019) propose a mechanism to allow a sensing system's central processor to make decisions based on minimal information from the collaborative network. Implementing this strategy in the context of our own project will reduce data transmission and help us save costs especially on the cloud side of our data pipeline. Another work, by Yu et al. (2013), proposes an incentivizing reputation mechanism to mitigate the potential presence of selfish users in the collaborative network–an important consideration for any system including ours which caters to outside users/general public. These principles are echoed in collaborative game-theoretic mechanisms–such as the idea of proof-of-work/state used in blockchain–which can be

used to incentivize cooperation and discourage exploitation in anonymous distributed systems. These considerations are especially relevant in the case of users who would like to build and maintain public versions of our pipeline technology.

Finally, Rajendran et al. (2018) built Electrosense, a platform that exposes spectrum data to the public domain through a collaborative monitoring network. The Electrosense system comes close to our project vision as a competitor, however it can still pose some accessibility and cost limitations especially for individuals and smaller organizations, and it is overall more focused on making raw data accessible, rather than additionally providing data processing solutions as our pipeline does.

3.4 Impacts from Preliminary Research

Ultimately, based on our examination of existing work in the realms of Spectrum Sensing and Collaborative Sensing Networks, we have incorporated several key ideas into our project. First, we built an affordable spectrum sensor using an RTL-SDR radio device connected to a Raspberry Pi microcomputer or personal laptop computer. This cheap and simple-to-make "DIY" sensor can be used to gather nearby spectrum signals which are later fed as input to our data pipeline.

Next, existing literature in this area has explored extensively the Collaborative Sensing Network, and we have used this model as the basis of our sensing system design–allowing for multiple users and sensors spread across geographical regions to all send signals to a centralized data pipeline which makes intelligent decisions regarding processing and ensuring the smooth flow of information.

4 Final Design – Details & Design Choices

4.1 Materials & Key Terms

For this project, there are a few Hardware Materials (Figure 2), several Software Materials for the General Pipeline (Figure 3), and other Software Materials specifically for certain parts of the design (Figure 4).

Hardware Materials	Description
SDR (Software Defined Radio)	Used to sense and collect raw radio spectrum data which is fed into the pipeline for future processing.
PC (Personal Computer)	Examples: Laptop, Desktop, Raspberry Pi. A type of computer designed for individual use with a processing unit that runs an operating system like Windows, macOS, or Linux, and can be used for a variety of tasks such as running software applications.

Figure 2: Key terms of Hardware Materials used

Software Materials	Description
Kubernetes	an open-source platform that automates the deployment, scaling, and management of containerized applications across multiple hosts and cloud platforms.
Docker	an open-source platform for developing, shipping, and running applications and can package an application and its dependencies into a single container that can be easily deployed and run on any platform/infrastructure.
AWS (Amazon Web Services)	a comprehensive cloud platform where we have built out the different stages and subsystems of our pipeline.
Amazon EC2 (Elastic Compute Cloud)	a web service in AWS that that allows users to rent virtual machines on which they can run their own applications.
Amazon EKS (Elastic Kubernetes Service)	a cloud-managed version of Kubernetes.
NATS (Neural Autonomic Transport System)	a high-performance connective technology that can manage data flow in distributed systems while ensuring efficient and reliable communication.

Figure 3: Key terms of Software Materials used in designing General Pipeline

Software Materials	Description	Design Section
Amazon S3 (Simple Storage Service)	a cloud-based object storage service that enables users to store and retrieve data from anywhere on the web.	Annotator
ES (Elastic Search)	a distributed search and analytics engine that provides database-like functionality to store, search, and/or analyze real-time data.	Search and Storage Engine
Flask	a micro web framework that provides the functionality to handle HTTP requests, process data, and generate dynamic responses to be sent back to the client-side of a web application.	Backend Web Server
Mapbox GL	a mapping API service used to create interactive maps.	Frontend
React	a Javascript library for building user interfaces.	Frontend
AWS Amplify	a frontend development framework and toolchain provided by AWS.	Frontend
Grafana	visualizes the system metrics and health of the entire pipeline.	System health Dashboard
Prometheus	a monitoring system and time-based database.	System health Dashboard
Docusaurus	a framework for easily building, deploying, and maintaining documentation websites.	Docs & Tutorials

Figure 4: Key terms of additional Software Materials

4.2 Details & Design Choices

The General Pipeline consists of 5 different components: Client, Annotator, Search & Storage Engine, Backend Web Server, and the Frontend Application. We additionally provide a System Health dashboard and Documentation/Tutorials. For our example Airplane Tracker Application, when a spectrum radio data packet is first processed by our Software-Defined Radio it will start being processed through the first five components until the corresponding airplane location appears as a point in the frontend application. The system health dashboard will monitor the pipeline in real time and the documentation will allow users to create their own application on top of our general pipeline. In this section, we will discuss the implementation detail and design choices for each of the 7 components.

4.2.a General Pipeline Design: Client Program

As the very first step, users will set up their radio data receiver (e.g. RTL-SDR (Realtek Software-Defined Radio)) connected to a Raspberry Pi microcomputer or personal computer. Based on the desired radio information, the user will select the proper frequency, bandwidth, gain, and other parameters of the device, and then the RTL-SDR will pass on the information in a raw digital format for further processing with software. The client program will be customized for the specific data to decode and process the information from the signal.

For our example Airplane Tracker, our RTL-SDR device will constantly listen and report the ADS-B (Automatic Dependent Surveillance-Broadcast) signals from airplanes in flight. This signal is an ADS-B packet transmitted by an aircraft in hexadecimal format, and can be seen in Figure 5. Each packet includes a 24-bit ICAO address (”A2FCF2” in figure) and a data payload (”1D59F0D8A109CCE2A2D6A3” in figure) which is later unpacked in our data pipeline and contains information about the aircraft’s position, altitude, and other relevant fields.

	Example Value	Description
ADS-B Packet	"A2FCF21D59F0D8A1-09CCE2A2D6A3"	Example of an ADS-B packet transmitted by an aircraft, in hexadecimal format.

Figure 5: ADS-B Packet

The client program will process the raw ADS-B packet into a readable format by using the dump1090 software which processes and decodes the data into a more human-readable format, in this case a JSON file. The client program returns this human-readable data as displayed in table-format in Figure 6 to be used in the next step of our pipeline. Now we have easy access to the airplane’s ICAO identification number, longitude, latitude and height.

Field Name	Example Value	Description
reporter	"7661c2aa-de08-477f-bfcb-8a54dc7991c9"	Unique identifier assigned to each user device reading & adding Spectrum data into our pipeline
reporter_uid	"josh-airspy"	Name identifier for each user device, configurable by user
time	"2023-05-01T22:50:50.-520538+00:00"	Timestamp at which Client program receives the broadcast data
ICAO	"A2FCF2"	Unique alphanumeric code assigned to aircraft by International Civil Aviation Organization (ICAO)
feet	13875.0	Current altitude of aircraft
lat	37.72631	Current latitude position of aircraft
lon	-122.48621	Current longitude position of aircraft

Figure 6: Data packet after processing by Client program

The Client is lightweight as it doesn't need to worry about other compute-intensive tasks such as annotating, ordering and processing of the data, which makes it easily deployable to a single board computer or embedded device such as Raspberry Pi. Also, these characteristics make it possible to deploy the Client multiple devices at the same time. So far we have tested up to 5 devices sending signals at the same time (distributed between California and Colorado). The setup of the Client is also straightforward with the help of our tutorial and multiple users have successfully set up their Clients using our documentation.

4.2.b General Pipeline Design: Annotator

The Annotator is a module within our Data Pipeline that is responsible for enriching the processed radio data with additional meaningful information. In a user's customized pipeline, this is the stage where they can supplement the incoming radio data with additional descriptive information that they would like for their application. For example, if a user wanted to make a music application and was reading in music-related radio data which contained the name of the song and artist, they may wish to append information from other data sources to each packet from available data sources–such as the number of listeners, days on top charts, etc.

In our Airplane Tracker application example, after the radio data broadcast by airplanes is sensed and fed into the NATS pipeline by the Client, it is passed to the Annotator, which then programmatically adds in important information necessary to create an airplane tracker application. For our application, we downloaded data from the FAA's (Federal Aviation Administration) Releasable Aircraft Database. This database contains information on the nation's registered aircraft that the Annotator uses to annotate the bare broadcast data it receives as input with the goal to add the manufacturer, model, unique United States aircraft identifier, and aircraft's name to each individual packet which we can later display on our frontend application.

Figure 7 is a screenshot of data items from our pipeline that have been processed by the Annotator module. Each JSON entry contains fields with specific characteristics of airplanes as described above like the ICAO number, manufacturer, registration, and location info–which has all been extracted or populated by the Annotator module in our pipeline.

Field Name	Example Value	Description
reporter*	"8e09d5b7-3413-43b2-9c3b-6feefb18182d"	Unique identifier assigned to each user device reading & adding Spectrum data into our pipeline
reporter_uid*	"sahai"	Name identifier for each user device, configurable by user
time*	"2023-05-01T22:50:50.520538+00:00"	Timestamp at which Client program running on user device has received the broadcast data
ICAO*	"A2FCF2"	Unique alphanumeric code assigned to aircraft by International Civil Aviation Organization (ICAO)
feet*	13875.0	Current altitude of aircraft
lat*	37.72631	Current latitude position of aircraft
lon*	-122.48621	Current longitude position of aircraft
manufacturer†	"BOEING"	Aircraft manufacturer
aircraft†	"737-900ER"	Aircraft model
n-number†	"N292AK"	Unique identifier assigned to aircraft by US Federal Aviation Administration (FAA)
registered†	"ALASKA AIRLINES INC"	Name of the aircraft's official registrant
annotator†	"00000000-0000-0000-0000-0e4f8c97f703"	Unique identifier assigned to the Annotator based on the machine/node annotator is running on, in our pipeline.

Figure 7: A detailed look into a single packet after it is processed by Annotator module

4.2.c General Pipeline Design: Search & Storage Engine

ElasticSearch (ES) is a distributed search and analytics engine that provides database-like functionality to store, search, and/or analyze real-time data. This is a core component in our pipeline that comes sequentially after the Annotator. It consumes all the annotated data from the pipeline and serves multiple purposes–including monitoring the Client status, storing the data, and finally serving as a persisten database to the backend flask server.

ES has the power to store a large amount of time-based log-like data, and its great scalability power allows its service to be highly-available and robust. We utilize ES to store 7-day spectrum data flowing through the pipeline for three main reasons: 1) ES allows us to control the rate of data flow from the annotator to the web server since this data is often not consistent in rate and can overwhelm the web server, causing failure. 2) Storing the data for any research analysis or debugging purposes. 3) For future steps, we want to enable Machine Learning integrations, which requires the data to be stored for training and test data.

Building out early iterations of our system played a significant role in helping us better refine our design choices and core values for our product. Initially, instead of using ES we were using MySQL for the backend persistent layer. MySQL is a relational database management system (RDBMS) that follows strict rules to ensure the integrity of the data inside as the topmost priority. However, we found that with the increase of data size and data issuing rate, over time, the MySQL capacity could not catch up and became a bottleneck in the pipeline. To address this problem, we turned to ElasticSearch (ES), a fast, scalable open search and analytics solution for time-based log-like data. A key insight in this transition is that since our collaborative sensing network contains multiple spectrum sensors often providing repeated and close data points, it is therefore crucial for our pipeline to be able to handle large amounts of incoming data within short time spans. ES is just designed for dealing with this scenario, which will ultimately add to the performance of our overall system. With a modest trade-off with data integrity, we improve the speed of data indexing by 5 times using ES in lieu of MySQL. Moreover, ES has rich add-ons such as machine learning integrations, which can be used to easily build up a machine learning service on top of the data inside the ES.

To allow ES to perform at its full capacity while also suiting the needs of our system, we employed a series of performance tuning modifications to ES. For example, we divided the data into multiple indexes based on read date, and set up an index template for these data to have a dedicated lifecycle policy. We divide the temperature of index data into the hot, warm and delete (cold) stages. As per the policy, each index will be in the hot stage at its creation, in which we will instruct ES to assign multiple replicas for it to accelerate the read rate. After a fixed amount of days, the index will go into the cold state, where the write operation to it is forbidden but the read operation is still guaranteed. Finally, it will be deleted after a month to preserve storage space on our server.

4.2.d Backend Web Server

The Backend Web Server's purpose is to fetch data using a query of preference from ElasticSearch, and pass the resulting data to the frontend application. We chose to use the Python micro-web framework Flask for the Backend Web Server since Flask is simple and easily integrates with other features. The Flask web server also provides multiple API endpoints for the web application to access the data. By designing a lightweight and customizable web server, we provide a framework that users can easily modify or extend to serve different applications.

For our Airplane Tracker, the Backend Web Server fetches the airplane data from ElasticSearch and passes it to the frontend Airplane Tracker display. The Backend Web Server is able to query with a specific time interval from which to fetch and serve data. The Flask Backend Web Server provides API endpoints for the Airplane Tracker Web Application to access. For example, one API endpoint allows the frontend to request airplane data between selected intervals. When the web server receives a request from the frontend, it determines what action to take based on the endpoint specified. Based on this specification the web server calls external services like ElasticSearch to fetch the required data. After that, the web server modifies the data in the correct format and sends the response to the frontend application.

4.2.e Frontend Application

The Frontend component functions to display the completely processed data from the pipeline in a specific form or application which the user had in mind for visualizing/utilizing the radio spectrum data. When a visitor interacts with the frontend web application, the application sends an HTTP request to the Flask web server. As described above, the web server in turn queries ElasticSearch, and finally sends the relevant information back to the frontend, where the user interface is updated accordingly.

For the frontend component of the pipeline, when a user is creating their own customized version it would be very unique to the data and application they are pursuing. Therefore this is one section of the pipeline that will most

likely have to be modified significantly by the user, but we have included details on our application so whichever parts are relevant can be used.

For our Airplane Tracker Application we aimed to have an interactive map with live airplane locations based on data read from our spectrum data collection devices. In alignment with that goal we had three subgoals to attain which are a) a website that can be accessed by anyone with internet access, b) a easy to understand map visualization so people can easily see plane trajectories and also click and learn more about each plane, and finally c) allow users to select a data and time range to see airplane flight trajectories from the past if needed. To achieve three goals we used a) AWS Amplify, b) React and Mapbox GL, and c) a third-party library, react-datetimerange-picker.

4.2.f System Health Dashboard

To monitor our pipeline, we built a system health dashboard for users and system administrators to check the pipeline status. We decided to use a combination of two tools– Prometheus and Grafana–since they work well together and are easily integrable into the rest of the NATS framework. Prometheus is a monitoring system and time-based database, which is paired with Grafana for graphical analysis of the collected metrics. Once we decided on the necessary technologies, we started working on connecting and enabling the flow of information between NATS, Prometheus, and Grafana. We used the Prometheus NATS exporter to collect metrics from NATS and expose them to Prometheus. From there, Prometheus is able to periodically scrape metrics and data from the exporter and stores the information in a time-series database. Finally, Grafana visualizes the metrics collected by Prometheus on the dashboard for users to access.

4.2.g Documentation

A core aspect of our project is the documentation and tutorials we provide, to allow others to replicate and tune our work for their own use cases. Our aim was for the documentation to be available on a publicly-accessible website, and there are many open-source documentation building tools to accomplish this–such as Sphinx, Gitbook, Docusaurus and Hugo. We chose to utilize the Docusaurus platform to write and deploy our documentation site because it has a simple and intuitive setup process that can help users like ourselves quickly create and deploy documentation websites. Also, Docusaurus generates static HTML pages, which makes the site quick to load and easy to cache. This can result in better performance in comparison to dynamic sites like GitBook. Last but not least, Docusaurus is backed by Facebook and has a large and active community of contributors, which means it receives frequent updates, bug fixes, and improvements.

5 Deliverables

5.1 System Health Dashboard

Since our pipeline contains several components and pieces that work together in conjunction, it becomes complex to monitor the system and debug any failures. Therefore, our comprehensive system health dashboard (as shown in Figure 8) is essential in allowing us to monitor the pipeline health in one central location through various metrics and visualizations. Specifically, we use the dashboard to detect various critical problems–such as excessive pending messages, storage overuse, potential packet loss, etc. Ultimately, the dashboard ensures the quality of observability for our pipeline, and is crucial as it entails the creator or operator's ability to monitor the health of the overall system.

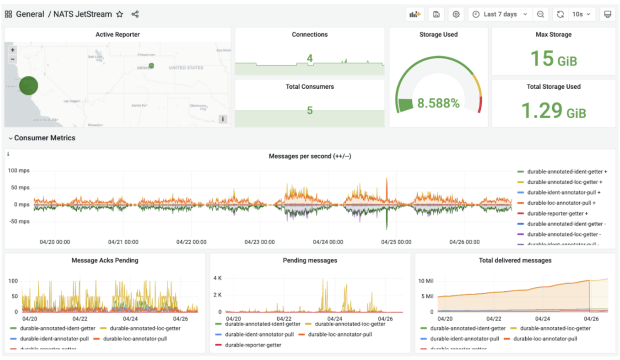

Figure 8: the System Health Dashboard

5.2 Documentation & Tutorials

As mentioned earlier, a major dimension of our project is to improve accessibility of spectrum data and processing solutions for the general public, and in order to achieve that goal we developed detailed documentation to allow users to replicate and build their own pipelines in a similar fashion to ours.

Our documentation includes detailed steps to build our pipeline for the Airplane Tracker example and guides users through the customization process to build their own pipeline. There are also debugging guides throughout, describing common issues we faced throughout the process. Our final set of documentation and tutorials can be viewed through a simple web URL and will appear as shown in Figure 9.

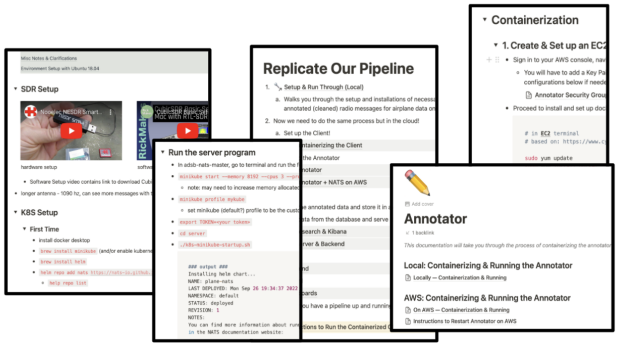

Figure 9: Screenshots from our Documentation website

5.3 Frontend

The Frontend refers to the public web application for our Airplane Tracker, which showcases the data from the spectrum resources in a way that is understandable for viewers. The web application opens to a map where airplane markers are updated in real-time every 20 seconds according to the actual locations being broadcast by planes currently in flight. Each airplane dot/marker on the map represents a logged location at a certain time, with a trail of the same color markers representing a single plane. By clicking on any marker, a popup will display additional information such as the airplane's ICAO number, altitude, time of location broadcast, the aircraft type, etc., as can be seen in Figure 10 below.

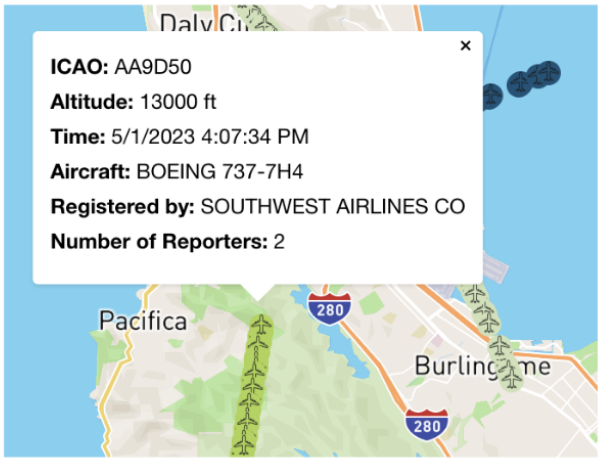

Figure 10: Popup information on a specific plane

As in Figure 10, there can be multiple reporters that detect the same broadcast by one plane. To distinguish between different reporters, the web application has a "filter by reporter" functionality, so users can only see markers detected by one specific reporter of their interest, shown in 11.

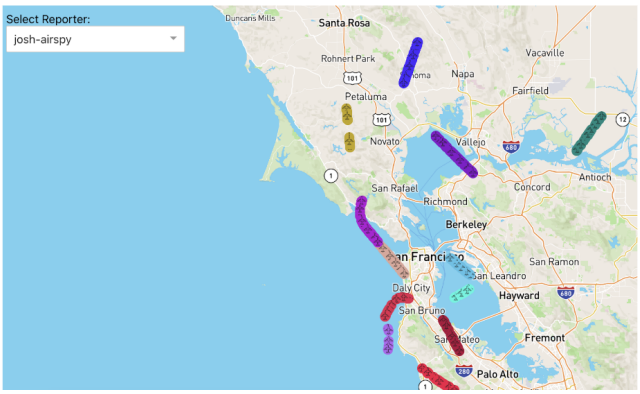

Figure 11: Airplane Tracker showing markers of one specific reporter

While the web application automatically starts showing planes in real-time, there is also the option to select a desired interval and see the plane trajectories during that time range in the top-left corner of the application. In Figure 12, we specified the start time to be 12:00 pm on Mar. 15, 2023 and the end time to be 12:04 pm on Mar. 15, 2023. Currently, we have limited the time range to be within 5 mins to avoid fetching tremendous amounts of data.

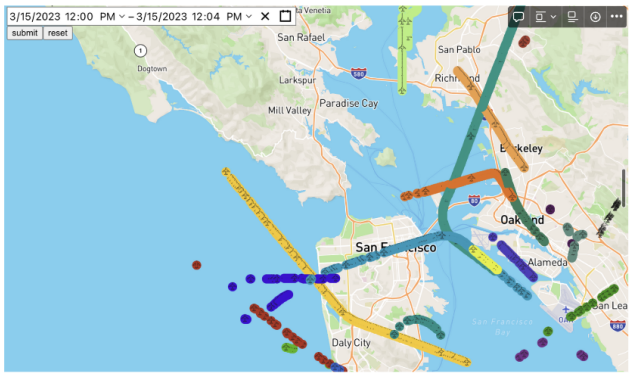

Figure 12: The Airplane Tracker application with many real-time plane trajectories

6 Evaluation

6.1 Stress Testing

6.1.a Purpose of Stress Testing

Stress testing is a type of testing used to evaluate the stability and robustness of a system or application under high-stress conditions, such as heavy load or limited resources. The purpose of stress testing is to identify and

address any weaknesses or vulnerabilities in the system that could lead to crashes, failures, or other performance issues. By simulating extreme conditions, stress testing helps ensure that the system can handle unexpected scenarios and maintains its functionality and performance.

6.1.b Metrics of Success for Stress Testing

We measured the metrics by using a local laptop as a client and publishing mock data to the pipeline at a constant speed. It is achieved by skipping the RTL-SDR device and sending human-readable airplane data to the pipeline directly. Since we used a shared queue for each client to translate and send messages, we used the exact same data structure in the stress testing script so that it can most accurately replicate the real use case of radio spectrum data being broadcasted, inserted into the pipeline, and removed from the queue of data after being processed.

The efficacy of our pipeline is determined by two essential components. The initial factor is the latency, where we strive to optimize the time taken by the pipeline to process incoming data. This metric is determined by calculating the time difference between the message's transmission from the client and the time it is received by the Frontend application. The second element is the throughput, which gauges the quantity of data that our pipeline can handle concurrently. To quantify this, we count the number of messages that are processed by the Frontend application. We have tested our system with various numbers of incoming messages per second to better understand how our pipeline reacts to different data flow rates in terms of latency and throughput.

6.1.c Stress Testing Results: Key Trends, Associations, Pipeline

Figure 13 is a graphical representation of the latency for different numbers of messages sent by the client, per second. The blue line represents the latency before an optimization while the orange line represents after an optimization. The optimization made to improve the latency to reasonable levels was to decrease the batch size and timeout so that the latency can be largely improved while maintaining a stable system. For the airplane tracker application, the number of messages per second is usually less than 50 so the latency for this application is mostly less than 1.4s.

Figure 14 is a graphical representation of the throughput for different numbers of messages sent by the client per second. The throughput compares the number of messages seen by the application versus the number of messages sent by the client. The blue line is our pipeline's throughput representation while the orange dashed line is the theoretical representation with slope being 1 since

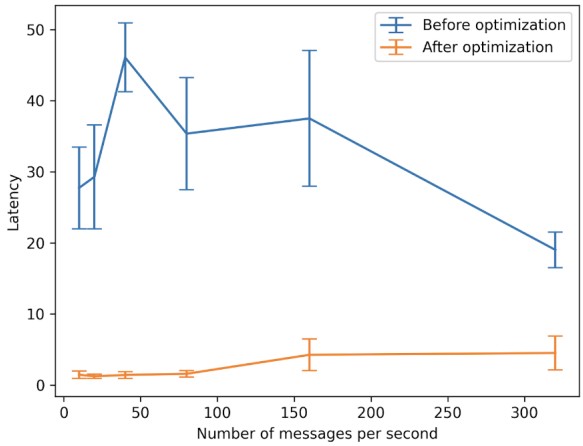

Figure 13: Latency

all messages should be delivered in time no matter how many messages are sent. Our system achieves peak performance when the number of messages is 800/s or below, which is far beyond the requirement for our current Airplane Tracker example application, but we can see that past that rate, our pipeline is not able to scale to full efficiency.

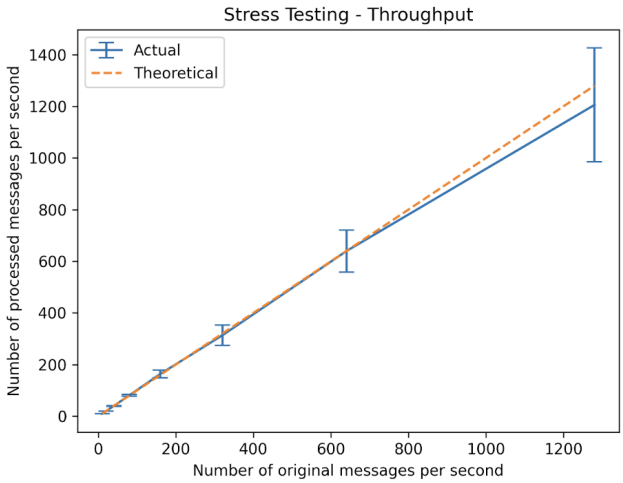

Figure 14: Throughput

6.2 User Study with Tutorials

In order to achieve our goal of enabling others to create their own end-to-end pipelines with spectrum resources, user studies to test the validity of our documentation and tutorials are essential. Our assessment involved collaborating with a user located in Colorado. This assessment was to test our instructions for adding an additional radio collection device to our Airplane Tracker pipeline. The user was provided with detailed documentation in-

structing them on how to use their own software-defined-radio (SDR) to collect and add plane information from their location in Colorado. This assessment was a success as the user could follow the instructions without needing additional assistance. We are now able to see airplanes in our Frontend in the Colorado areas as well as shown in Figure 15 below.

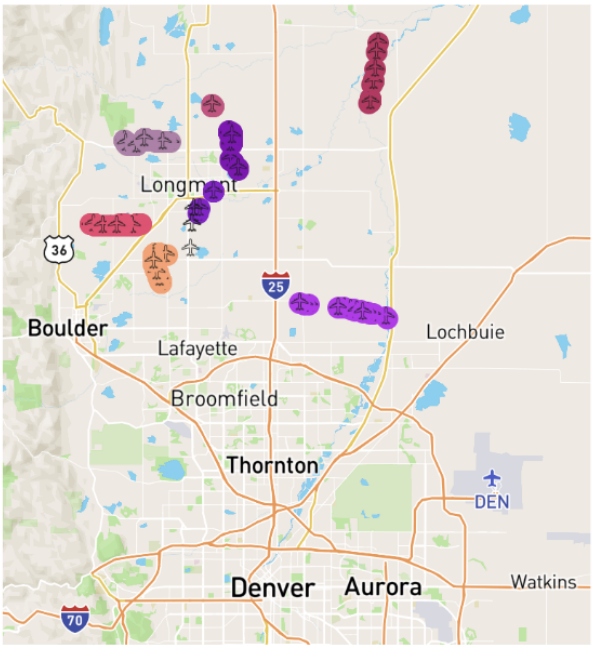

Figure 15: Airplane Tracker data in the Colorado region

This user study highlighted the importance of creating easy-to-use documentation and by following up with additional user studies targeting users of various levels of experience, we can continue to refine and improve the quality and cohesiveness of our documentation.

7 Conclusion

7.1 Ultimate Significance of Results

Our final product proves that an end-to-end pipeline can be created in an affordable and accessible manner without the need for large-scale resources. Our stress testing has proved that this pipeline is robust, lightweight, and portable, to allow users to efficiently process and analyze spectrum data. Our user study for our tutorials has proven that our tutorials are detailed and effective enough for others to understand and follow.

This research represents a significant contribution to the field of spectrum data processing and analysis. Our pipeline has the potential to democratize access to spectrum data and enable a broader range of stakeholders to leverage this valuable resource for their own purposes.

7.2 Implications for Future Work & Potential Next Steps

Based on our assessments, we propose several directions for future work with a focus on our data pipeline design and the user experience.

First, we suggest additional user studies and testing to validate the effectiveness of our tutorials for users with varied experience levels. This can help identify any potential areas for improvement and refine the tutorial for optimal usability by diverse audiences. There are also several Machine Learning (ML) algorithms that can be utilized to improve our pipeline. For example, ML can be utilized for an anomaly detection task to filter out potential malicious or inaccurate data. Elasticsearch provides us with ML APIs to enable this, allowing for the creation of custom layers on top of our current pipeline.

Furthermore, when processing high-throughput data at scale, there are limitations to what messaging systems like NATS can handle on their own. Alternative messaging frameworks like ZeroMQ or nanomsg can address the scaling needs as they offer advanced features like proxy servers that can help to alleviate the limitations of NATS. By using proxies, data can be transferred between nodes without the need for the central messaging system to be involved in every transaction, which can significantly reduce the load on the system. To make use of this advanced functionality while still leveraging the benefits of NATS, the solution is to use ZeroMQ or nanomsg orchestrated via control messages sent over NATS. By doing so, the system can benefit from the scalability and performance of these messaging systems while still preserving the reliability and message delivery guarantees of NATS.

These proposed directions for future work have significant implications for the field of spectrum data processing and analysis. The refinement of data pipeline design and user accessibility can facilitate the development of more effective and user-friendly applications in various domains such as telecommunications and security.

8 References

Bavaskar, Pranita P., Omkar Kemker, and Aditya Kumar Sinha. "A Survey On: 'Log Analysis With Elk Stack Tool'." International Journal of Research and Analytical Reviews 6, no. 4 (2019).

Bhat, Poojya J, and Priya D. "Modern Messaging Queues – Rabbitmq, Nats and Nats Streaming." International Journal of Recent Technology and Engineering (IJRTE) 9, no. 2 (2020): 402–8. https://doi.org/10.35940/ijrte.b3551.079220.

Gómez, Cristian Osvaldo. "Apuntes Azure." GoConqr. Accessed April 13, 2023. https://www.goconqr.com/note/23097218/apuntes-azure.

He, Shibo, Kun Shi, Chen Liu, Bicheng Guo, Jiming Chen, and Zhiguo Shi. "Collaborative Sensing in Internet of Things: A Comprehensive Survey." IEEE Communications Surveys & Tutorials 24, no. 3 (2022): 1435–74. https://doi.org/10.1109/comst.2022.3187138.

Kleber, Nikolaus, Abbas Termos, Gonzalo Martinez, John Merritt, Bertrand Hochwald, Jonathan Chisum, Aaron Striegel, and J. Nicholas Laneman. "RadioHound: A Pervasive Sensing Platform for Sub-6 GHz Dynamic Spectrum Monitoring." 2017 IEEE International Symposium on Dynamic Spectrum Access Networks (DySPAN), October 19, 2016. https://doi.org/10.1109/dyspan.2017.7920764.

R. Aurangzaib, W. Iqbal, M. Abdullah, F. Bukhari, F. Ullah, and A. Erradi, "Scalable Containerized Pipeline for Real-time Big Data Analytics," 2022 IEEE International Conference on Cloud Computing Technology and Science (CloudCom), Bangkok, Thailand, 2022, pp. 25-32, doi: 10.1109/CloudCom55334.2022.00014.

S. Rajendran et al., "Electrosense: Open and Big Spectrum Data," in IEEE Communications Magazine, vol. 56, no. 1, pp. 210-217, Jan. 2018, doi: 10.1109/MCOM.2017.1700200.

Saber, Mohammed, Hatim Kharraz Aroussi, Abdessamad El Rharras, and Rachid Saadane. "Raspberry Pi and RTL-SDR for Spectrum Sensing Based on FM Real Signals." 2018 6th International Conference on Multimedia Computing and Systems (ICMCS), 2018. https://doi.org/10.1109/icmcs.2018.8525867.

Smith, Peter J., Rajitha Senanayake, Pawel A. Dmochowski, and Jamie S. Evans. "Distributed Spectrum Sensing for Cognitive Radio Networks Based on the Sphericity Test." IEEE Transactions on Communications 67, no. 3 (2019): 1831–44. https://doi.org/10.1109/tcomm.2018.2880902.

T, Sharvari & K, Sowmya. (2019). A study on Modern Messaging Systems- Kafka, RabbitMQ, and NATS Streaming.

Yu, Chung-Kai, Mihaela van der Schaar, and Ali H. Sayed. "Distributed Spectrum Sensing in the Presence of Selfish Users." 2013 5th IEEE International Workshop on Computational Advances in Multi-Sensor Adaptive Processing (CAMSAP), 2013. https://doi.org/10.1109/camsap.2013.6714090.