311 CHICAGO SERVICE REQUESTS DATA PROCESSING PIPELINE REPORT

Susmitha Marripalapu (A20489531)
Prajwal Premdas (A20473680)
Tinh Cao (A20476426)

CS554 Big Data Technology 05/04/2022

1 PROJECT TOPIC - 9

Create a big data processing pipeline Explore the use of tools such as Kafka or AWS Kinesis and AWS DynamoDB or EMR to accept and process data in real-time or "simulated" real-time.

2 PROJECT MILESTONES

- Collecting the dataset. - All - 04/03/2022
- Get data from source and upload to S3 bucket - Susmitha - 04/03/2022
- Create Lambda function which cleans data and pushes to another S3 bucket - Susmitha - 04/06/2022
- Created an Elastic Container Registry to create an image with pandas, NumPy, and shapely as we are using these python packages for data curation - Susmitha - 04/06/2022
- Create DynamoDB Table - Create using the data schema - 1 - Tinh - 04/07/2022
- Create Lambda function to process cleaned data to DynamoDB - watch the ProcessedData bucket and upload the new data to DynamoDB. - Tinh - 04/12/2022
- Configure Kinesis data stream for DynamoDB - Prajwal - 04/20/2022
- Configure Kinesis Firehose - Prajwal - 04/21/2022
- Configure QuickSight dashboard for data visualization and analysis - All - 04/26/2022
- Prepare content for the report - All - 05/01/2022

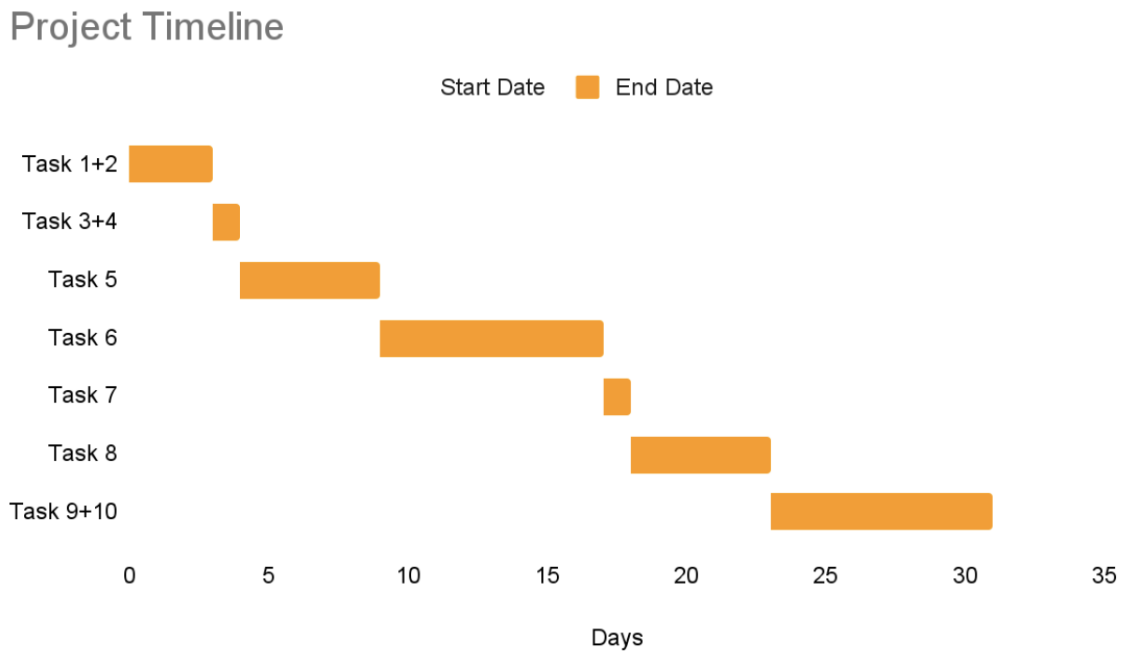


Fig : Timelines

3 LITERATURE OVERVIEW

3.1 AWS S3

Amazon Simple Storage Service (Amazon S3) is an object storage service offering industry-leading scalability, data availability, security, and performance. With cost-effective storage classes and easy-to-use management features, you can optimize costs, organize data, and configure fine-tuned access controls to meet specific business, organizational, and compliance requirements.
In the project, we get the raw data and store it in an S3 bucket. We watch this bucket constantly and process the data if anything new is added. This is done using AWS Lambda.
The processed data is stored in another bucket from where it is loaded to DynamoDB using a AWS Lambda function.

3.2 AWS LAMBDA

AWS Lambda enables one to run code without provisioning and managing servers. Polling, checkpointing, and error handling complexities are abstracted, leaving the user to solely focus on business logic. When Kinesis data stream are subscribed with AWS Lambda functions, AWS Lambda automatically read batches of records from the data stream and process them if records are detected on the stream. AWS Lambda periodically polls the stream (once per second) for new records and when it detects new records, it invokes the Lambda function passing the new records as parameters. The Lambda function is only run when new records are detected.
We have created two python Lambda functions to process the data. One to watch the S3 bucket, clean the data and upload the processed data to another S3 bucket . Another to load processed data from the S3 Bucket to DynamoDB.

3.3 AMAZON DYNAMODB OVERVIEW

DynamoDB is first introduced by Sivasubramanian (2012), an engineer working for Amazon, to eliminate the complexity and operational overhead of a seamlessly scalable database service. It is a fast, flexible NoSQL database service for single-digit millisecond performance at any scale. DynamoDB offers built-in security, continuous backups, automated multi-region replication, in-memory caching [6], and data export tools. Major principles of DynamoDB were influenced by the Dynamo Paper written in 2007 [5], whose project was developed by Amazon.com to achieve

reliability at a massive scale. The "always-on" experience principle was carried through from Dynamo Paper to DynamoDB, where consistency was sacrificed under certain failure scenarios to ensure a persistent state follows Brewer's CAP theorem [4], eventual consistency.

- **AMAZON DYNAMODB ARCHITECTURE:**
  For horizontal scalability, DynamoDB uses consistent hashing to write data into multiple data nodes and automatically increases the number of nodes in the backend when required to handle more data [6]. A good use case for DynamoDB would be an application with strict latency requirements, a serverless application, like AWS Lambda, communicating through HTTP API to event triggers, and simple known data set structure to elevate DynamoDB's key-value/document access patterns and relax JOIN operations from a relational database.
  Originally, Sivasubramanian (2012) introduces DynamoDB as a purpose-built service that targets non-relational databases, but the newest update notes from Jame (2022) [7] have seen DynamoDB shifting into the general-purpose territory, particularly SQL-like operational works. Starting in 2021, ADDG has added enhanced support for ACID transactions, replication size and availability, backup, streaming, and on-demand pricing to broader its integration with other AWS services like AWS S3, AWS Kinesis Streaming, and AWS CloudTrail. Regarding usability, AWS [6] also announced support for NoSQL Workbench - a client-side, cross-platform IDE tool for data modeling and visualizing - and PartiQL, where DynamoDB users can interact with data tables using common SQL commands. On December 1, 2022, Amazon DynamoDB introduces the new Amazon DynamoDB Standard-Infrequent Access (DynamoDB Standard-IA) table class, which reduces the infrequent access table storing cost by up to 60 percent [3]. Users can switch between the DynamoDB Standard table and DynamoDB Standard-IA class depending on storage requirements and data access patterns.

- **AMAZON DYNAMODB COMPETITORS:**
  Another technology that also competes with DynamoDB in this space is Amazon's DocumentDB. On the data architecture, DocumentDB uses JSON-Like document-based (semi-structured) to store, process, and retrieve data easily [11]. It's also fast, scalable, and highly available that is fully compatible with the MongoDB database service. Under availability, DocumentDB follows an immediate consistency model with Atomic single-document operations for transaction processing.

  Amazon Neptune (rank 108) by DB-Engines employs Graph Database and Resource Description Framework (RDF), where information is represented as triples in the form of subject-predicate-object [9]. Graph Database represents data in graph structures with nodes and edges, which are the relationship between them. The power lies in the properties of graph and graph traversal for text search. Since the technology used in Amazon Neptune does not integrate well due to compatibility issues and functional issues with aws real-time processing pipeline, Amazon Neptune will not be considered further in our report.

  DocumentDB does not provide sharding for storing different data on different nodes while DynamoDB does [12]. The data size for DocumentDB is smaller, limited to 16MB for document, and the storage is maximized at 64 TB of data. DynamoDB provides no limit on document scaling, 2MB partition key length, and the smallest item size is 400 KB [11]. DynamoDB employs both Document Store and Key-value Store, whereas DocumentDB only uses the Document model. With DocumentDB, engineers need to configure the infrastructure, such as the DocumentDB instance class, number of nodes, virtual private cloud (VPC) point, and authentication. After launches, DocumentDB starts managing its server and performs other operational tasks such as creating daily backups, upgrading engine versions, or increasing disk storage as needed [9]. In addition, stored procedures are required in DocumentDB for data retrieval and data accumulation when writing queries. In contrast, DynamoDB users can start creating the table instantly on-the-spot with minimal infrastructure configuration.

  The difference between DynamoDB and DocumentDB lies in the cost structure and operations [9]. DynamoDB works best when applications access and retrieve data based on key peruse cases, but works inefficiently for queries that scan over the entire table. Stored Procedures are not needed under DynamoDB as users can create query data queries directly to the database. For availability, DynamoDB automatically replicated global table across multiple AZs in a single region or regions if specified. Any failures can be easily mitigated by switching over to other replicas in different regions while waiting for the isolated Region to come back online. While if the primary node fails in DocumentDB, the services will read data from replicas to the primary

node and multi-AZ has to be configured explicitly for this to work [9]. On a cost basis, DynamoDB comes with on-demand pricing so users only pay for the time they run the application, which can even be less than $1.On the contrary, Document DB starts with an up front cost of 200$ per month regardless of time use or instance use, making it less attractive compared to DynamoDB.

3.4 STREAMING SOLUTION

- **APACHE KAFKA AND AMAZON KINESIS OVERVIEW:** Data streaming services validate and route messages from one application to another, managing message queues and workload effectively. With this, users will be able to process messages and handle large data streams more efficiently. We have many data stream services available for configuration and use. Two famous services which are widely used in organizations are Amazon Kinesis and Apache Kafka.

  Apache Kafka is an open-source "event streaming platform" — a platform that writes and reads event streams. Kafka handles data streams in real-time (like Kinesis.) It's used to read, store, and analyze streaming data and provides organizations with valuable data insights. Uber, for example, uses Kafka for business metrics related to ridesharing trips.
  Amazon Kinesis is a real-time data streaming service that captures data from various sources, including operating logs, social media feeds, website clickstreams, financial transactions, and more. Kinesis then processes and transforms this data and loads it into a data store for analytics. Netflix, for example, uses Kinesis to process billions of traffic flows.

- **APACHE KAFKA VS AMAZON KINESIS FUNCTIONALITY:** The big difference between Kinesis and Kafka lies in the architecture. Kafka "decouples" applications that produce streaming data (called "producers") in the platform's data store from applications that consume streaming data (called "consumers") in the platform's data store. Kafka has more of a scattered nature than Kinesis, making it useful for node failures.

  If messages should be kept for over 7 days, Kafka could provide a solution. However, it requires (lots of) human support to make the data stream process work. Kafka takes longer to set up than Kinesis. You'll need a team to install (and manage) data clusters. Kafka supports only Java whereas Kinesis (via AWS) supports Java, Go, Android, and .NET. Hence we choose not to use Kafka and use kinesis as a streaming service in this data processing pipeline.

- **KINESIS DATA STREAM ARCHITECTURE:** The units that construct the barebone of an AWS data stream are called shards. Shard represents a sequence of data records in a stream that comes from various IoT endpoints [1]. Currently, a shard is limited to 1 MB/second or 1,000 records per second for data ingestion and an outbound throughput (reads) of 2 MB/sec. Users can adjust the number of shards up or down to match the throughput capacity needed for their workload. Big companies like Netflix can easily incur thousands of shards in a single stream. The unique structure of shards allows for parallel data processing, with ordered defined based on the time it was stored. A unit of data stored in AWS Kinesis data stream is called a record, composed of sequence number, partition key, and data blob. The sequence number is the unique identifier that is automatically assigned when the data producer calls PutRecord or PutRecords API function to add data to a Kinesis data stream. The PutRecords operation sends multiple records to the data stream per HTTP request while PutRecord only sends one record per HTTP request. A partition key is created in a similar manner, where it's used to group records with the same key into the same shard. A data blob refers to the actual size of the raw data before being encoded and added into a data stream - the maximum size is 1 MB.

- **READING AND PROCESSING DATA FROM KINESIS DATA STREAMS:** One of the main advantages of Kinesis Data Streams processing is that it allows for multiple unique applications to consume data from the same Kinesis data stream at the time thanks to the introduction of enhanced fan-out technology [3]. Before the adoption of the technology, users needed to create multiple AWS Lambda functions, each dedicated to a unique program, that shares the same 2 MB/second outbound throughput. The shared bandwidth constraints create the bottlenecks in the number of lambda functions running concurrently, their performance, and the provision of more than one data stream to meet up with the demand. Using enhanced fan-out, now one stream with the same number of shards can scale the output to match the number of consumers, maximizing at five per shard.

3.5 INTERACTIVE QUERY SERVICES

- **AMAZON ATHENA:** Amazon Athena is an interactive query service that makes it easy to analyze data in Amazon S3 using standard SQL. Athena is serverless, so there is no infrastructure to manage, and you pay only for the queries that you run. Athena is easy to use. Simply point to your data in Amazon S3, define the schema, and start querying using standard SQL. Most results are delivered within seconds. With Athena, there's no need for complex ETL jobs to prepare your data for analysis. This makes it easy for anyone with SQL skills to quickly analyze large-scale datasets.

- **AMAZON ATHENA VS REDSHIFT SPECTRUM:** Amazon Redshift Spectrum is a feature within Amazon Web Services' Redshift data warehousing service that lets a data analyst conduct fast, complex analysis on objects stored on the AWS cloud. Essentially, both Athena and Redshift Spectrum do the same thing: query S3 using standard SQL, and store the results. There is only one major difference between Athena and Spectrum: Athena stores query results on S3, which can be loaded into Redshift from there; while Spectrum can join tables directly on Redshift.

  Redshift Spectrum can be considered if queries should be closely tied to a Redshift data warehouse. A Redshift table can be created by joining S3 data with Redshift data. Spectrum makes it easy to do this. If all the data is in S3, Athena is a better option. It is probably not worth the effort and cost to spin up a Redshift cluster just to use Spectrum if you are not looking to analyze Redshift data. Instead, use Athena to read from S3.

3.6 KINESIS FIREHOSE

The main usage for Kinesis Firehose is to batch streaming data into a persistent storage service like S3 or Elasticsearch. The serverless, auto-scaling, and ease of implementation make Kinesis Firehose a good candidate if our usage does not depend on real-time access to data.

3.7 VISUALIZATION TECHNOLOGY

Data Visualization software solutions have quickly become one of the most crucial tools in the market. Tableau is not the only Data Visualization software available, and it is not necessarily the perfect choice for all requirements. That is why we have compared Tableau with Quicksight, weighing one software against the other.

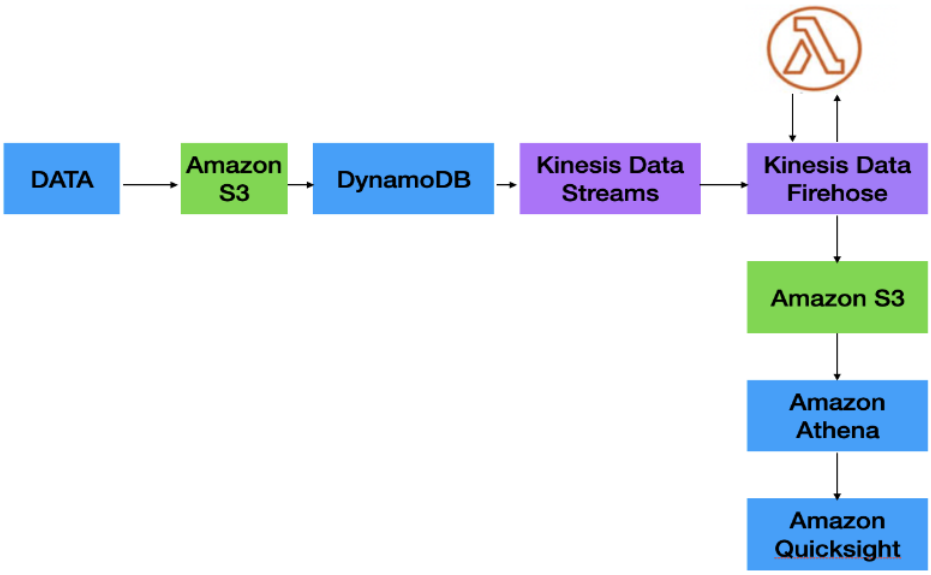
- **QUICKSIGHT AND TABLEAU OVERVIEW:** Amazon QuickSight is a fully managed cloud-scale business intelligence service that is powered by machine learning and allows you to create data visualizations and dashboards for the people you work with, wherever they are. It's a potential competitor to business intelligence tools like Tableau and Microsoft's Power BI.

  Tableau can help anyone see and understand their data. Connect to almost any database, drag and drop to create visualizations, and share with a click. Tableau provides features like Online Analytical Processing (OLAP), Analytics, Filtered Views, Dashboard Creation, Relational Display, etc. Whereas, Quicksight provides Performance Metrics, Ad hoc Analysis, Ad Hoc Reports, Dashboard, Data Analysis, and other such functionalities.

- **QUICKSIGHT VS TABLEAU:** Quicksight is a web-based BI tool. There are many BI tools like PowerBI, Tableau, etc. But, they need some specific system infrastructure. However, Quicksight doesn't need any system infrastructure. That is the main benefit of using Quicksight. When assessing the two solutions, we found Amazon QuickSight easier to use and set up. However, Tableau Server is easier to administer. Business analysts can seamlessly create serverless, pixel-perfect dashboards in minutes–securely, Developers can deploy and scale embedded analytics to 100s of 1000s of users in apps with robust AWS APIs, Admins can provide consistent performance as QuickSight automatically scales to the workload and End-users in organizations can ask questions in natural language and receive answers with relevant visualizations.

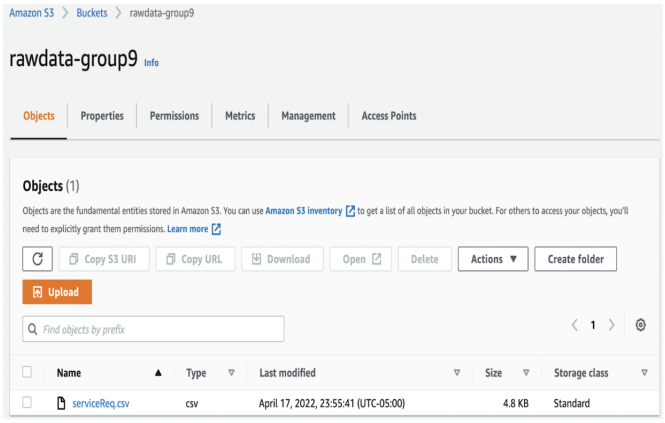
4 PROCEDURE

4.1 DATA PROCESSING PIPELINE ARCHITECTURE

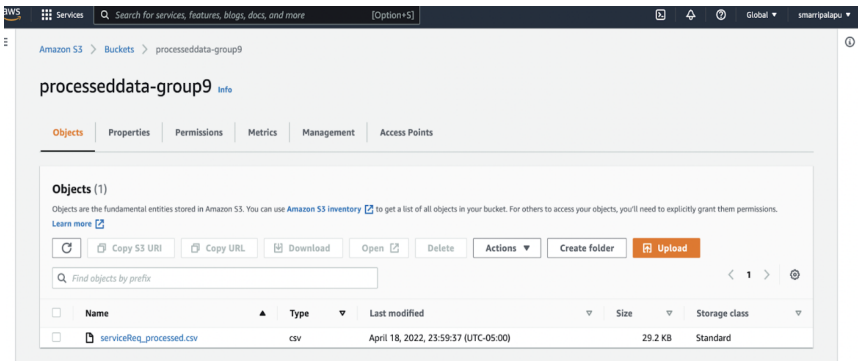


4.2 Creating Amazon s3 bucket to hold raw and processed data in S3.

S3 Bucket to hold raw 311 service requests data



S3 Bucket to hold processed 311 service requests data

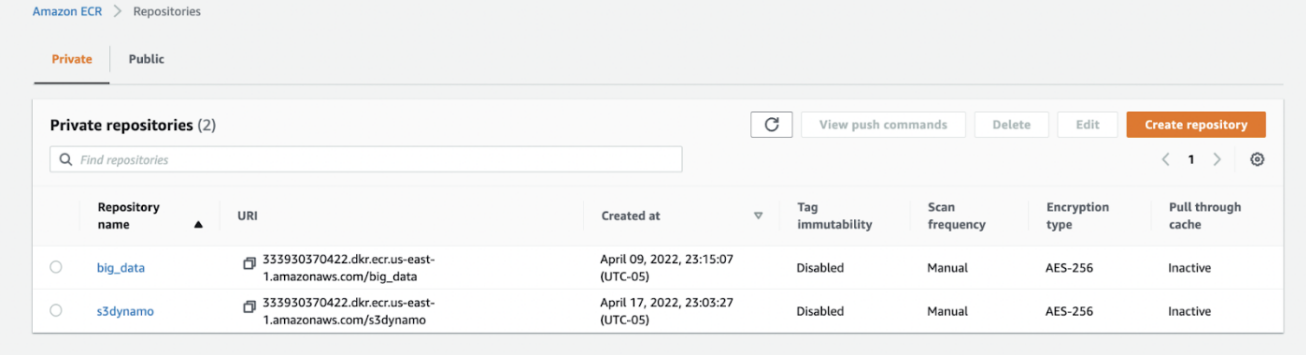


4.3 Create an ECR image using Dockerfile.

Since we need python packages like pandas, numpy and shapely for data curation, we created an image in Elastic container registry to hold these packages with the python script. The repository created for dataCurationFunc is

“big_data” as shown in the figure.

ECR Repository



Docker file code is as follows:

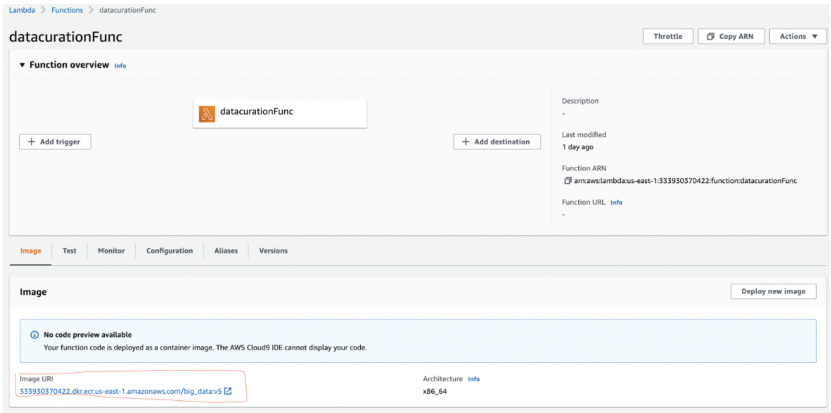
```
FROM public.ecr.aws/lambda/python:3.8
COPY app.py ${LAMBDA_TASK_ROOT}
RUN pip install numpy --target "${LAMBDA_TASK_ROOT}"
RUN pip install pandas --target "${LAMBDA_TASK_ROOT}"
RUN pip install shapely --target "${LAMBDA_TASK_ROOT}"
CMD ["app.handler"]
```

Build docker and push it to ECR Repository

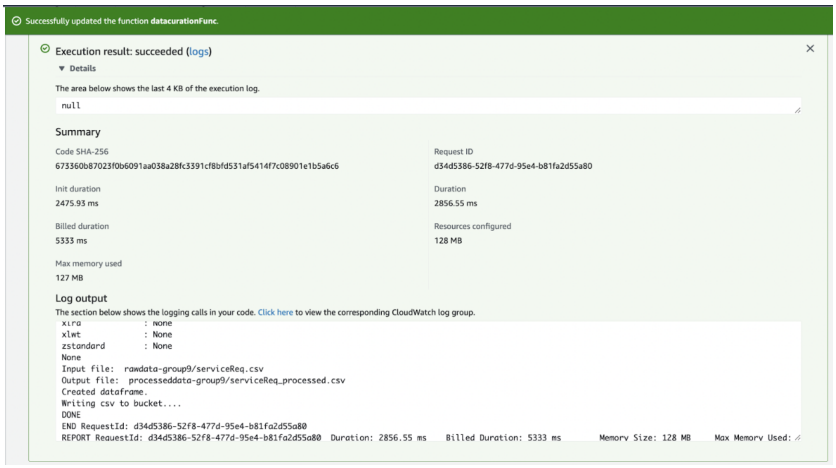
```
(base) susmitham@Susmithas-MBP pandas_lambda % docker build -t big_data .
[+] Building 18.4s (10/10) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 346B
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [internal] load metadata for public.ecr.aws/lambda/python:3.8
 => [internal] load build context
 => => transferring context: 189B
 => [1/5] FROM public.ecr.aws/lambda/python:3.8
 => [2/5] COPY app.py /var/task
 => [3/5] RUN pip install numpy
 => [4/5] RUN pip install pandas
 => [5/5] RUN pip install shapely
 => exporting to image
 => => exporting layers
 => => writing image sha256:1e98032bbb30866bda6099ad5e03c524b5e12fd8035779ced2fb92d112a46347
 => => naming to docker.io/library/big_data

Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to fix them
(base) susmitham@Susmithas-MBP pandas_lambda % docker tag big_data:latest 333930370422.dkr.ecr.us-east-1.amazonaws.com/big_data:latest
(base) susmitham@Susmithas-MBP pandas_lambda % docker push 333930370422.dkr.ecr.us-east-1.amazonaws.com/big_data:latest
The push refers to repository [333930370422.dkr.ecr.us-east-1.amazonaws.com/big_data]
2969a7a49cfb: Pushed
0436dbfd518d: Pushed
ce4b22275dd0: Pushed
437f6b9b87e9: Pushed
622a3c9a860b: Pushed
3049fc9ea433: Pushed
159cfb847db8: Pushed
69dee4b59c29: Pushed
d75fe6680705: Pushed
0df2d8c86f6a: Pushed
latest: digest: sha256:a30d819d872457a0080d56de409434c7dc4aaa1b3f537f3767277cb2aa806802 size: 2424
(base) susmitham@Susmithas-MBP pandas_lambda %
```

4.4 Create Lambda function for data curation and creation of processed data file in S3:

lambda function which executes on raw data for data cleaning.



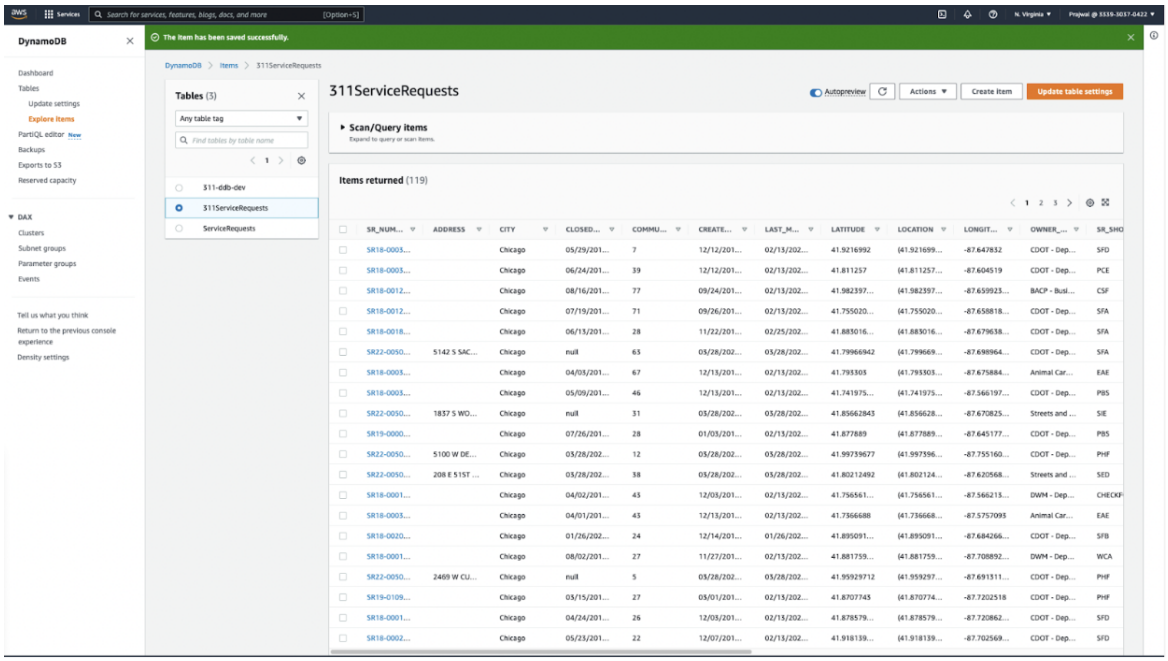
Execution result.



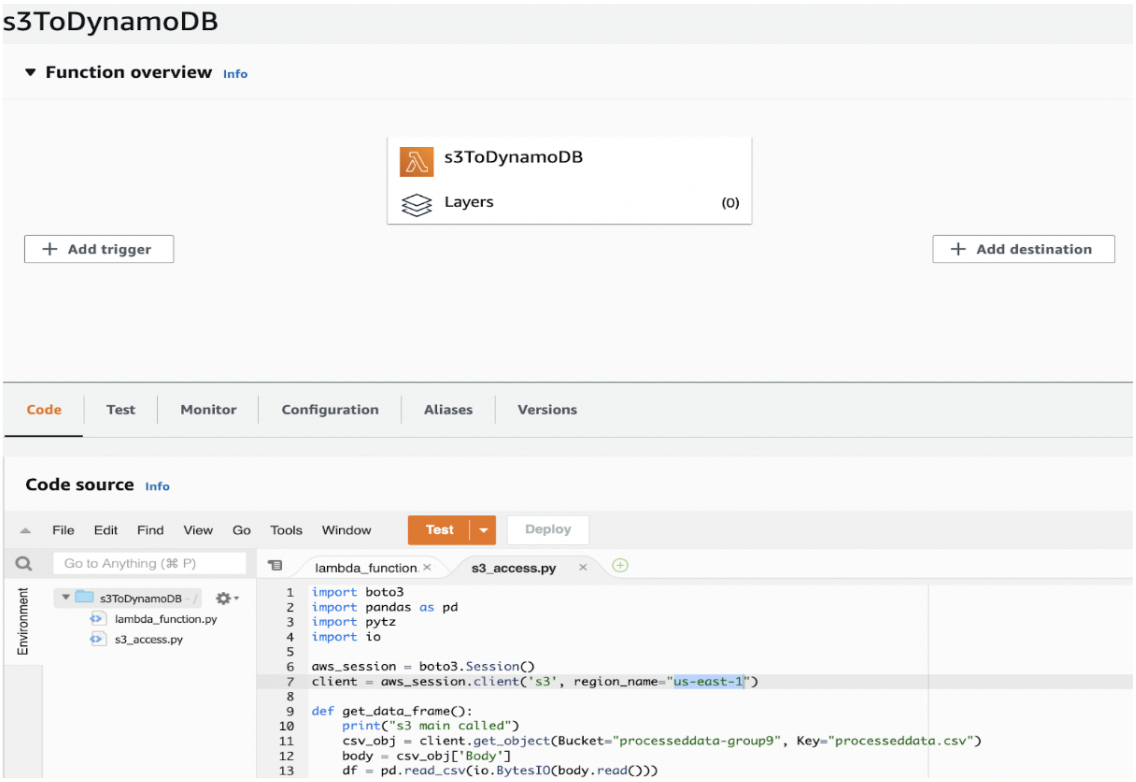
4.5 Data curation

Data curation helps us to access and look for useful information for analysis by creating, organizing and maintaining datasets. Cleaned data can be used to support business decision-making, academic needs, scientific research and other purposes. To make data consistant and reduce redundancy, we have written a logic to curate service requests data and use it for processing. For code refer this link: Datacuration code

4.6 Create Dynamodb table to hold data.



4.7 Created s3ToDynamoDB lambda function to transfer the data from processed data bucket to dynamodb



For code refer this link: s3ToDynamoDB code

4.8 Configure kinesis data stream.

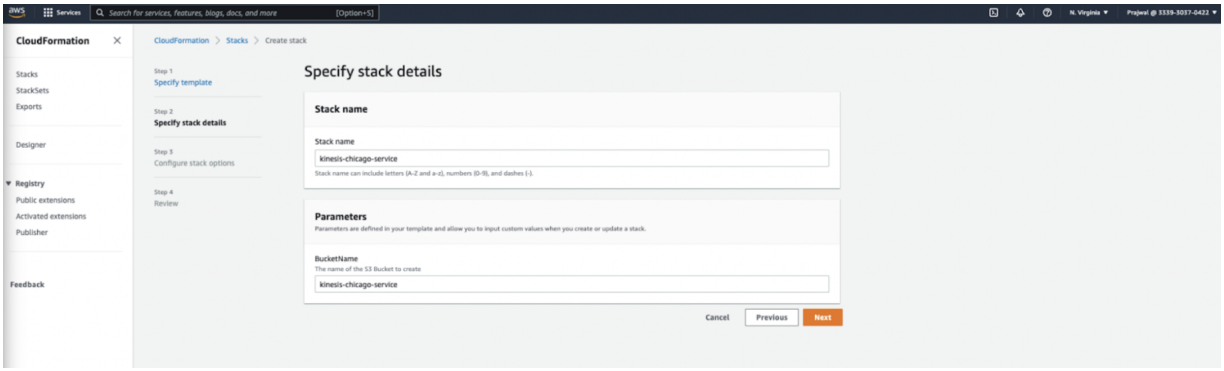


Fig: 4.8.1 Kinesis datastream stack setup

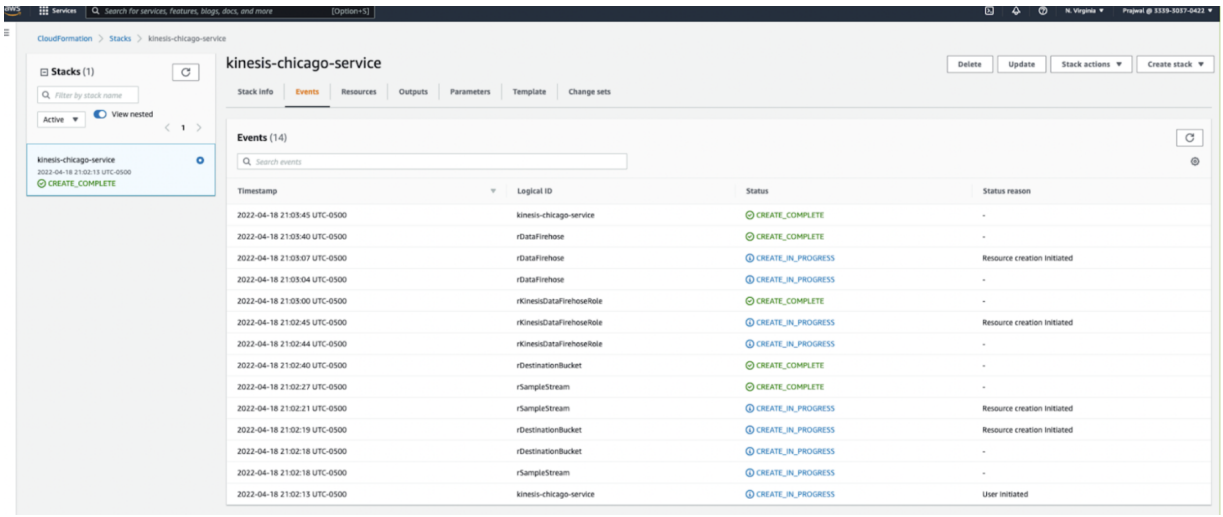


Fig: 4.8.2 Kinesis setup events

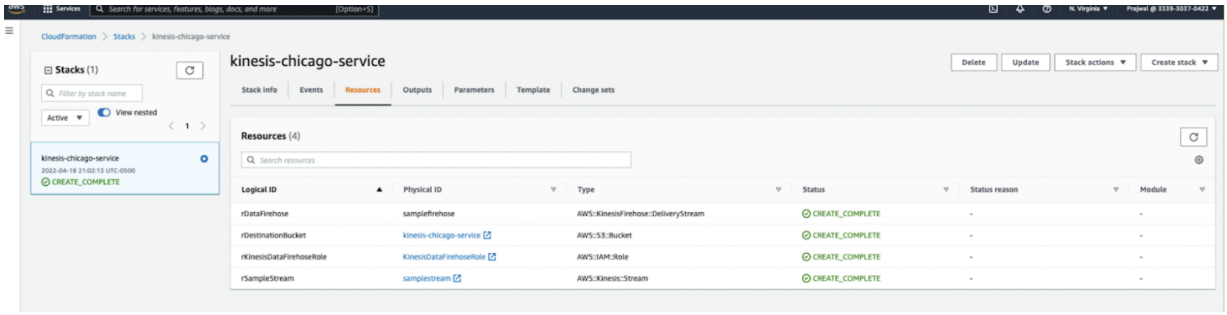


Fig: 4.8.3 Kinesis setup resources

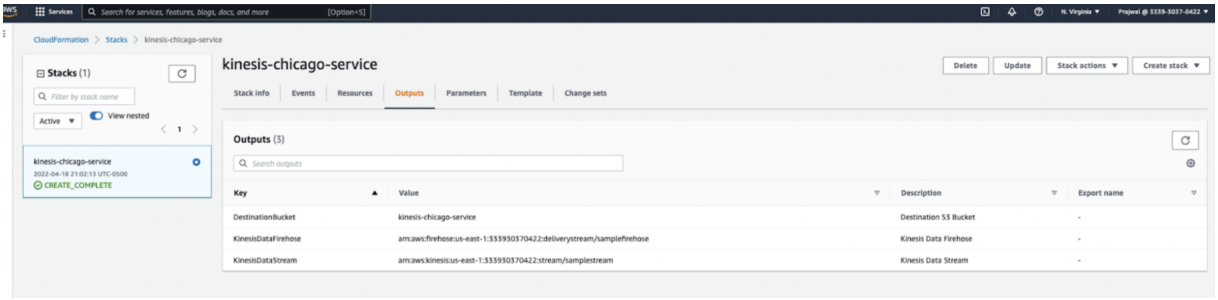


Fig: 4.8.4 Kinesis setup outputs

4.9 Enable kinesis datastream for DynamoDB:

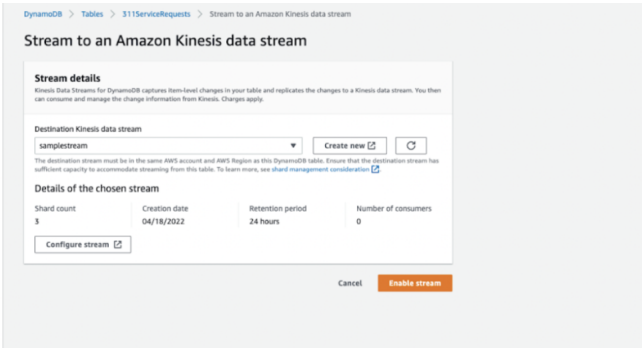


Fig 4.9.1 Enable kinesis datastream

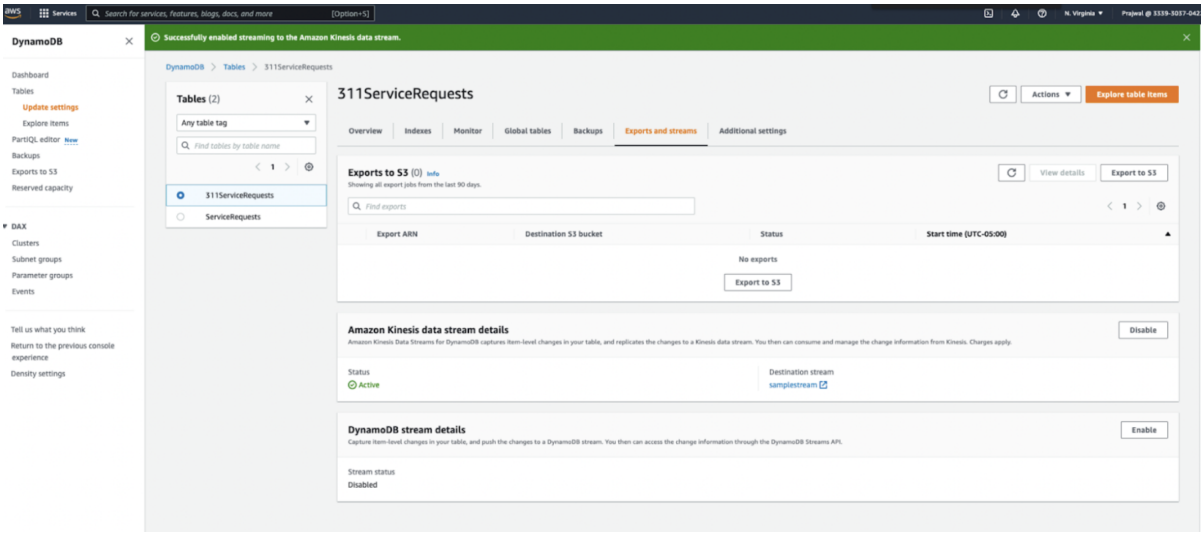


Fig 4.9.2 Check configurations in DynamoDB

4.10 Lambda function to transform data for kinesis firehose:

By default, Kinesis Data Firehose sends JSON records inline, which causes Athena to query only the first record in each S3 object. To overcome this, we use a Lambda function to transform records before sending them to Amazon S3 by adding an end of line (EOL) character.
For code, refer this link: athena_data_transform

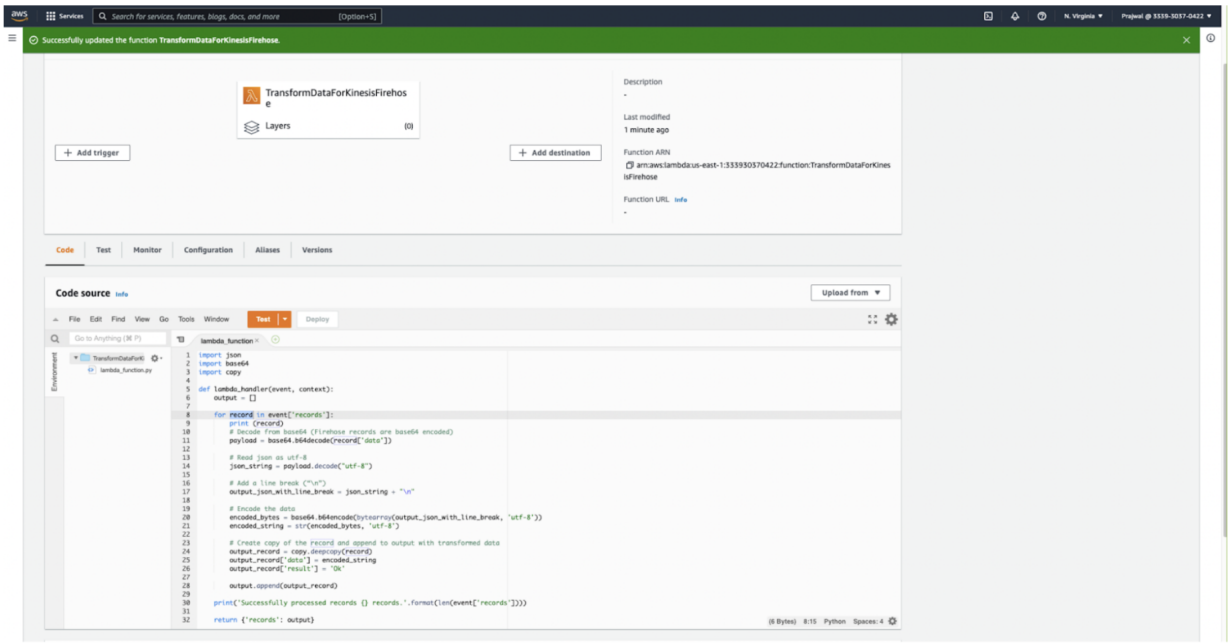


Fig 4.10.1 lambda function to transform data for kinesis firehose

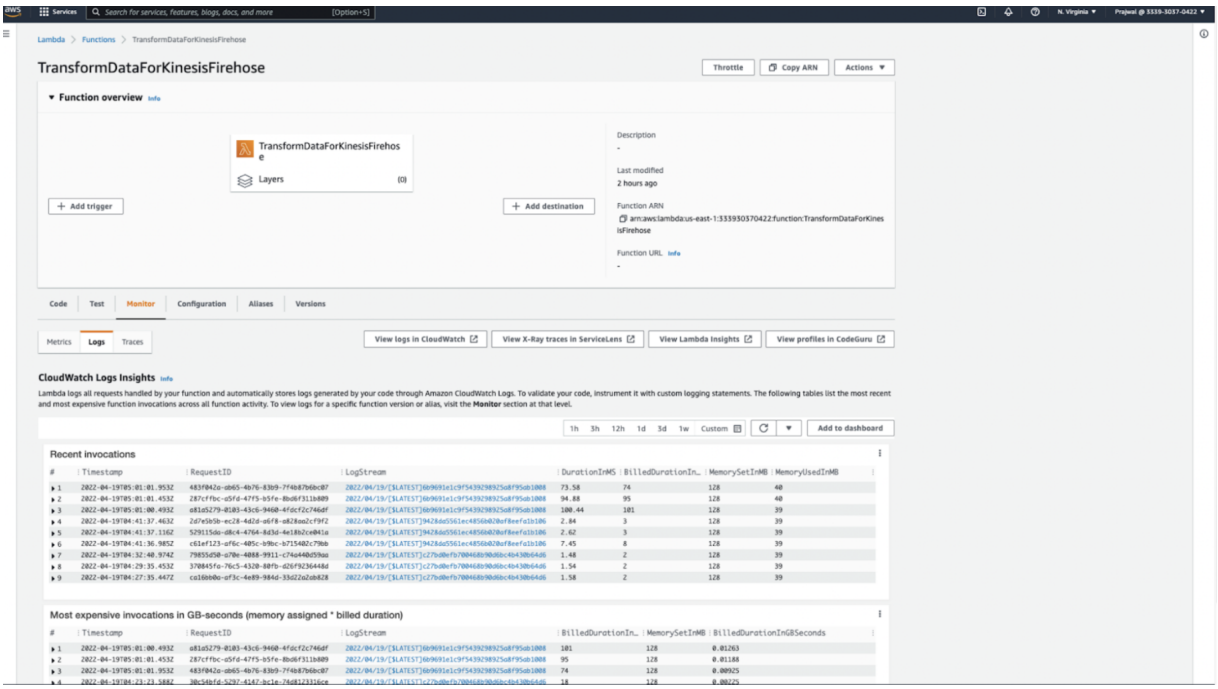
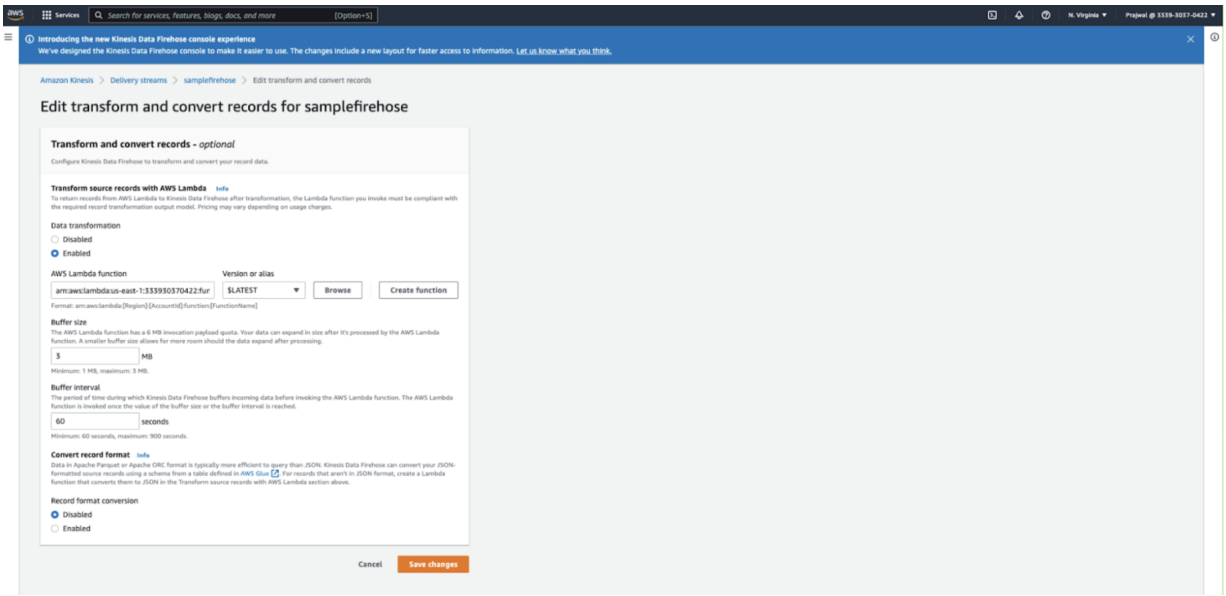


Fig 4.10.2 lambda function results

4.11 Kinesis firehose configuration for data stream:

Transform source records with AWS Lambda, for Source record transformation, select Enable.



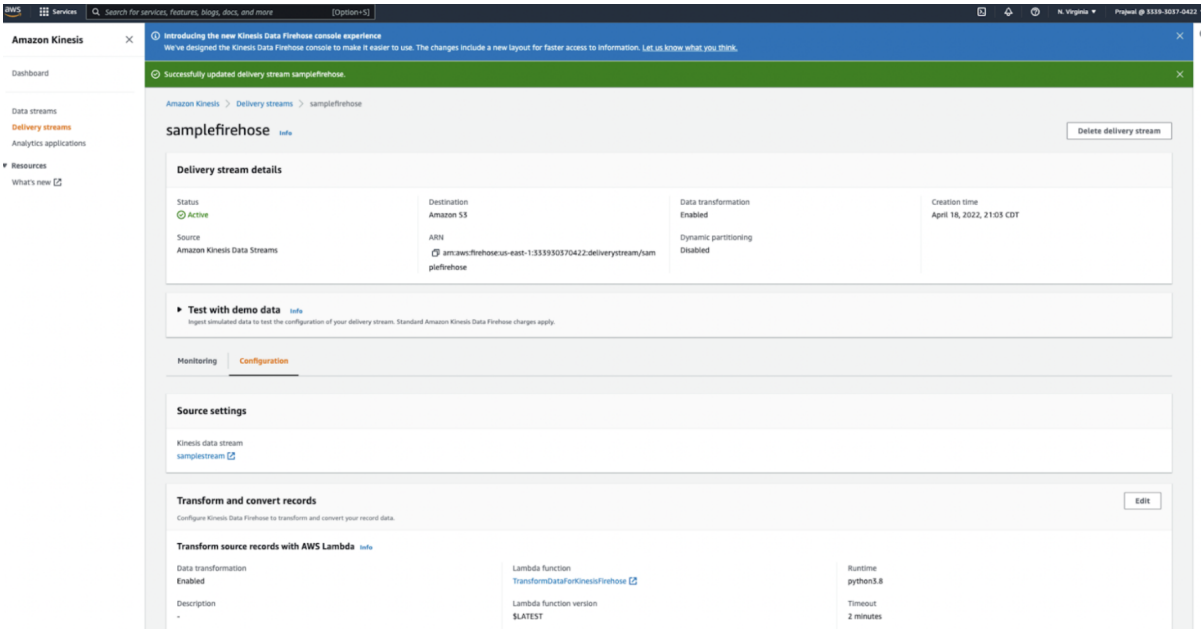


Fig 4.11.1 Kinesis firehose configuration

4.12 Configure Athena:

We can now create a database and table in Athena and query the data. Query to create database in Athena:

```
create database chicagoservice;
```

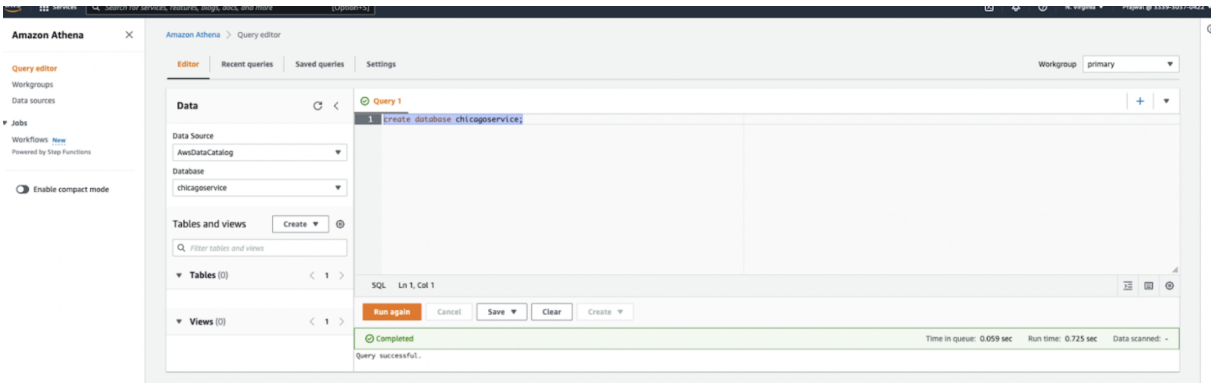


Fig 4.12.1 Create Database in Athena

Query to create table in Athena:

```
CREATE EXTERNAL TABLE servicerequests(
        dynamodb struct<NewImage:struct<
                LAST_MODIFIED_DATE:struct<S:string>,
            COMMUNITY_AREA :struct<N:integer>,
            CITY :struct<S:string>,
            STREET_TYPE :struct<S:string>,
            STREET_NUMBER :struct<N:integer>,
            CREATED_DATE :struct<S:string>,
            STREET_ADDRESS :struct<S:string>,
            CLOSED_DATE :struct<S:string>,
            SR_TYPE :struct<S:string>,
            STREET_DIRECTION :struct<S:string>,
            SR_SHORT_CODE :struct<S:string>,
```

```
            STATUS  : struct<S: string >,
            ADDRESS  : struct<S: string >,
            STREET_NAME  : struct<S: string >   ,
            LATITUDE  : struct<N: float >   ,
            STATE   : struct<S: string >   ,
            ZIP_CODE   : struct<N: integer >   ,
            LONGITUDE  : struct<N: float > ,
            OWNER_DEPARTMENT  : struct<S: string >
        > > )
ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe' WITH SERDEPROPERTIES
( 'ignore.malformed.json' = 'true') LOCATION 's3://kinesis-chicago-service/';
```

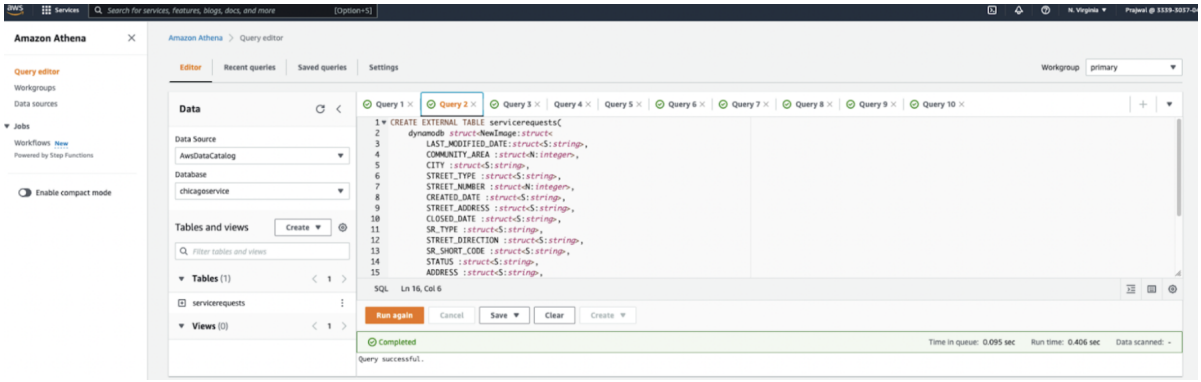


Fig 4.12.2 Create table in Athena

Query data from Athena table:

```
SELECT
dynamodb.NewImage.ZIP_CODE.N as ZipCode,
dynamodb.NewImage.STATE.S as State,
dynamodb.NewImage.CITY.S City,
dynamodb.NewImage.CREATED_DATE.S as CreatedDate,
dynamodb.NewImage.LATITUDE.N as Latitude,
dynamodb.NewImage.LONGITUDE.N as Longitude,
dynamodb.NewImage.OWNER_DEPARTMENT.S as Department,
dynamodb.NewImage.SR_SHORT_CODE.S as SR_Code,
dynamodb.NewImage.CLOSED_DATE.S as ClosedDate,
dynamodb.NewImage.STREET_NAME.S as StreetName,
dynamodb.NewImage.STATUS.S as Status,
dynamodb.NewImage.COMMUNITY_AREA.N as CommunityArea
FROM "chicagoservice"."servicerequests"
```

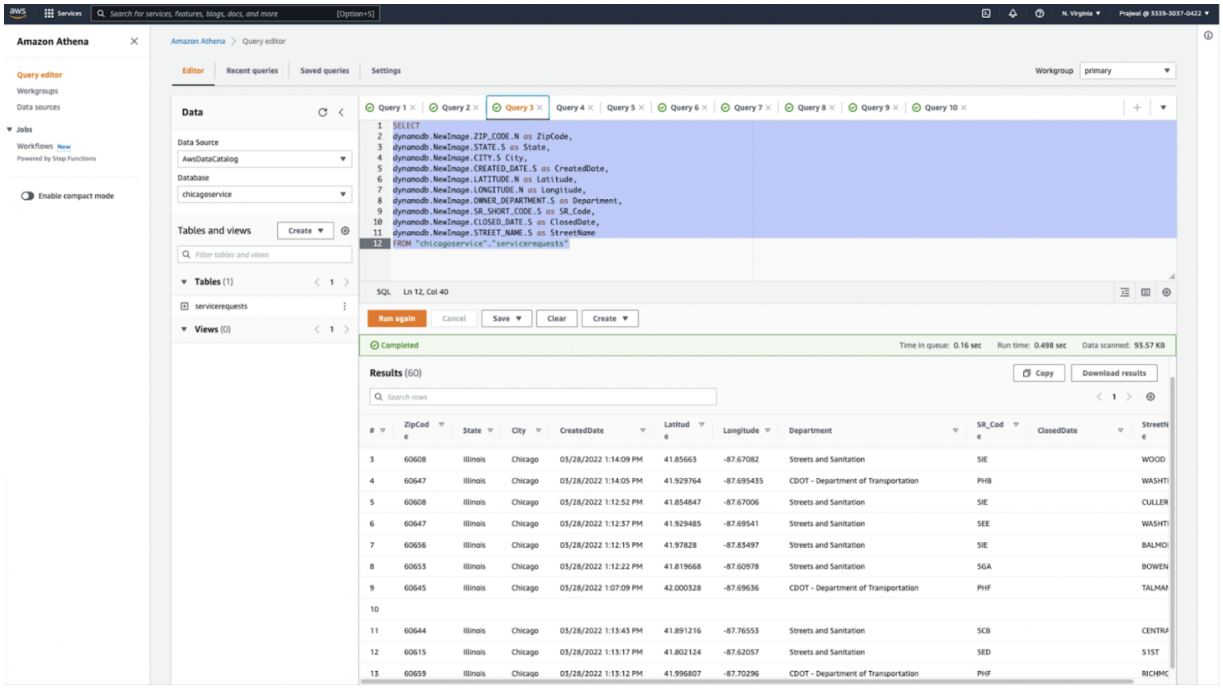


Fig 4.12.3 Query data

4.13 Data processing:

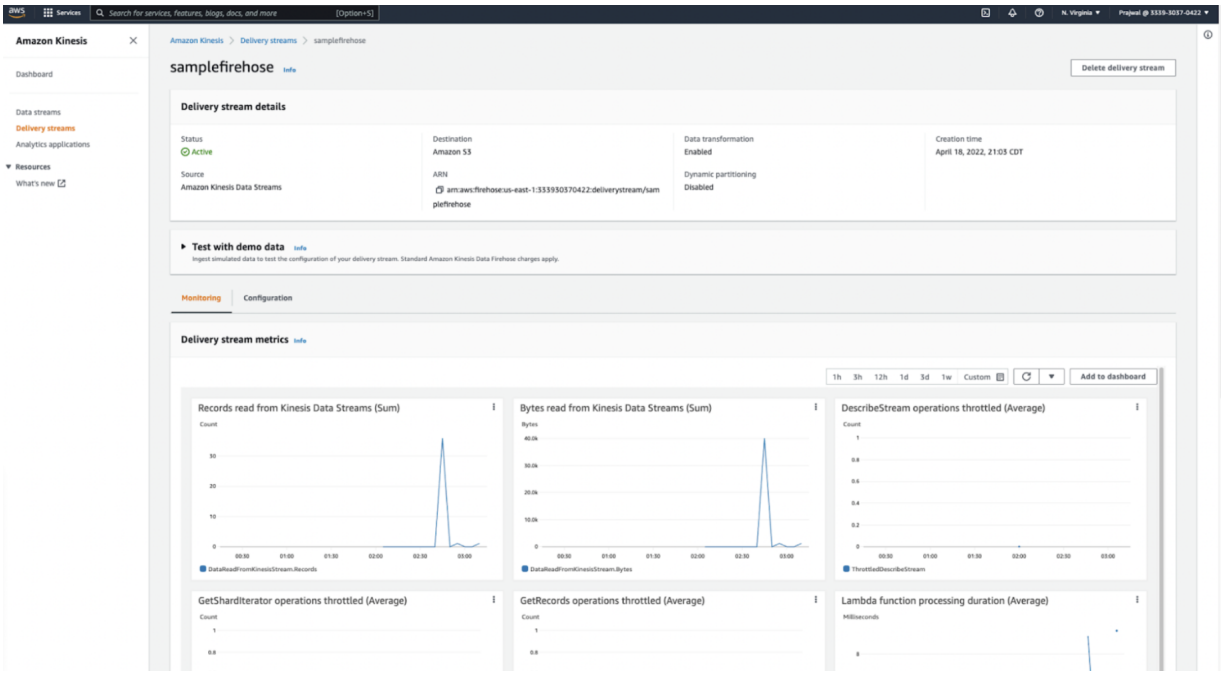


Fig 4.13.1 Firehose data processing metrics

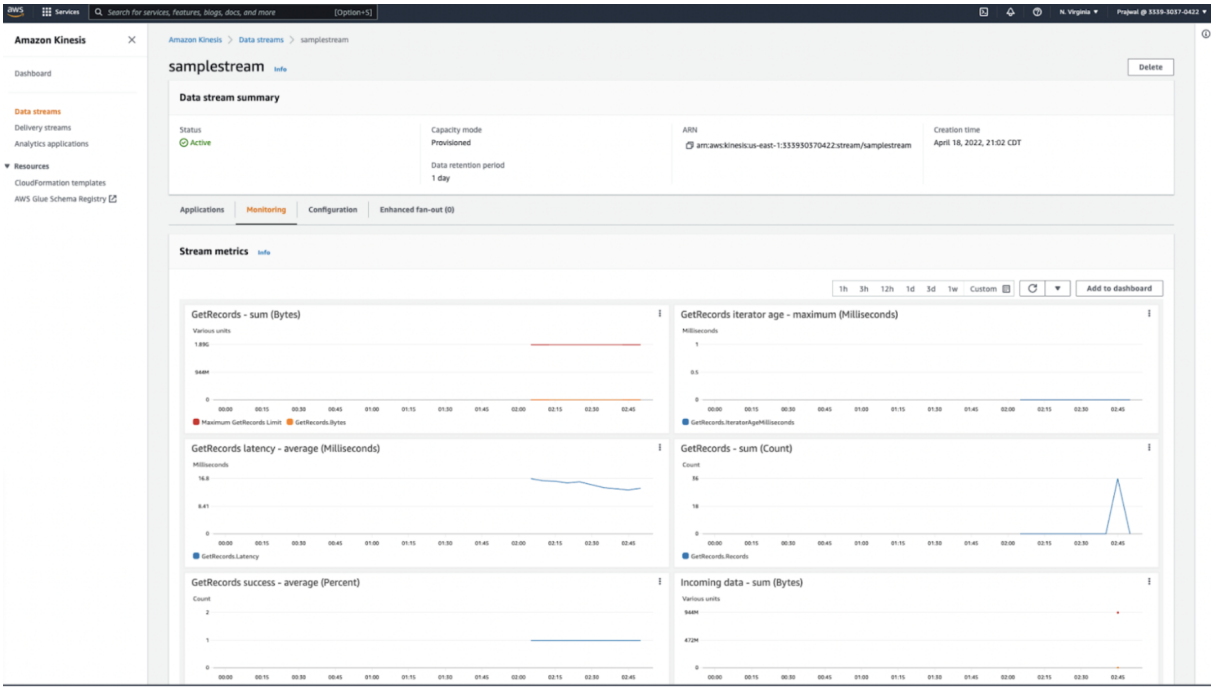


Fig 4.13.2 Kinesis stream metric

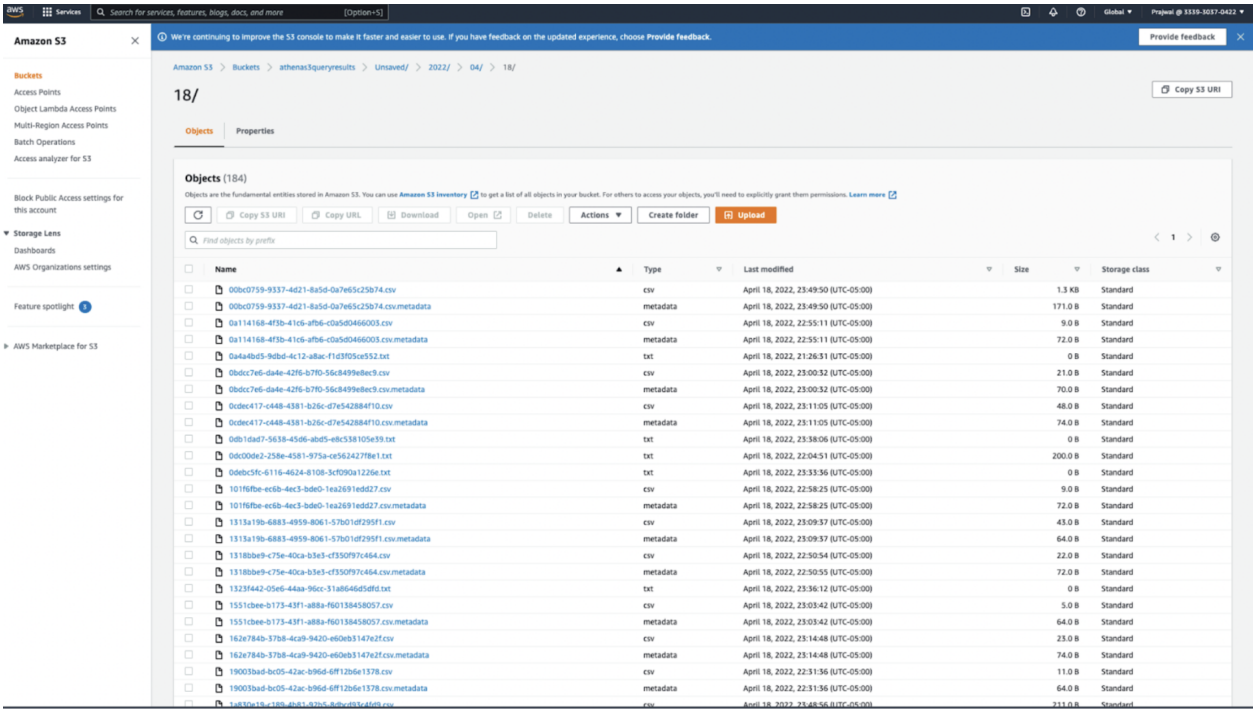


Fig 4.13.3 S3 buckets processed by Athena

4.14 Configure Quicksight:

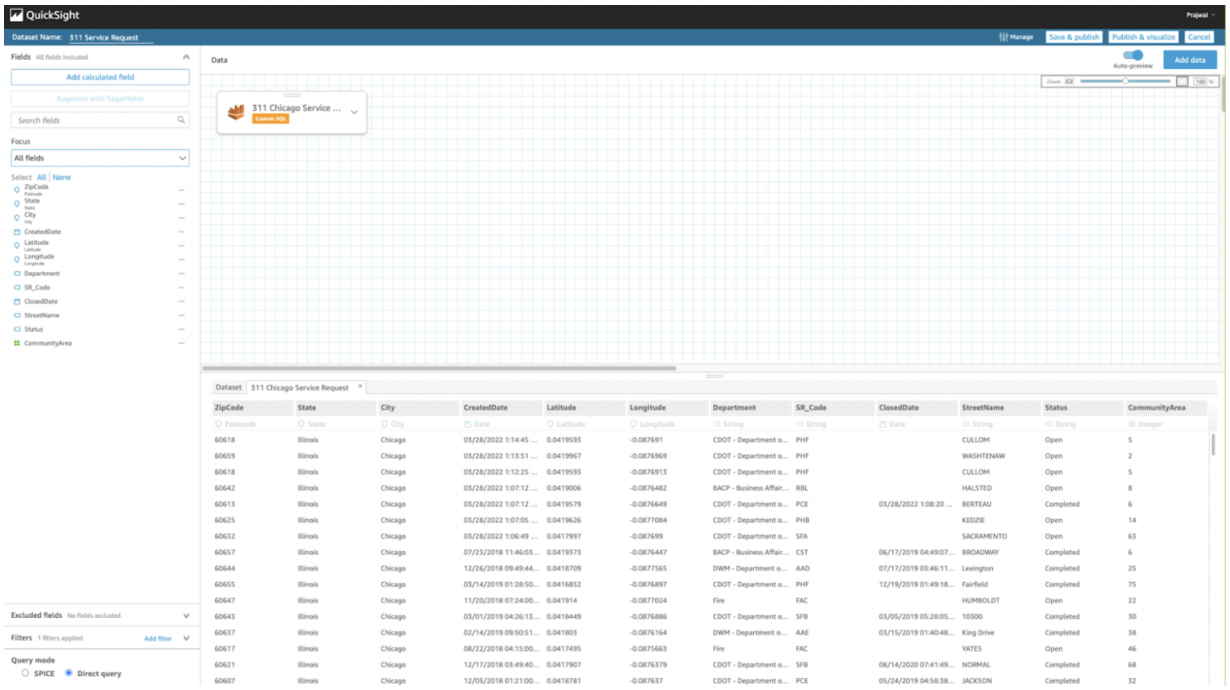


Fig 4.14.1 Data in Quicksight from Athena

4.14.1 VISUALIZATIONS

Heatmap to show all the services requested by the users in Chicago area.

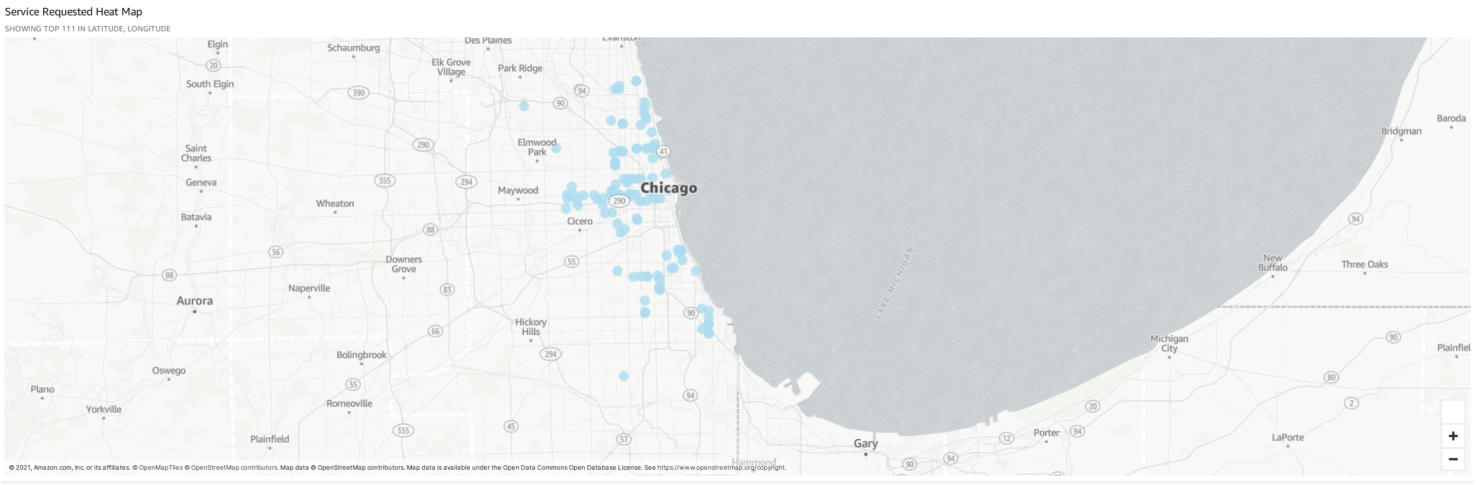


Fig 4.14.2 Heatmap visualization in quicksight

All the services request by the users daily and stacked by Street Name.

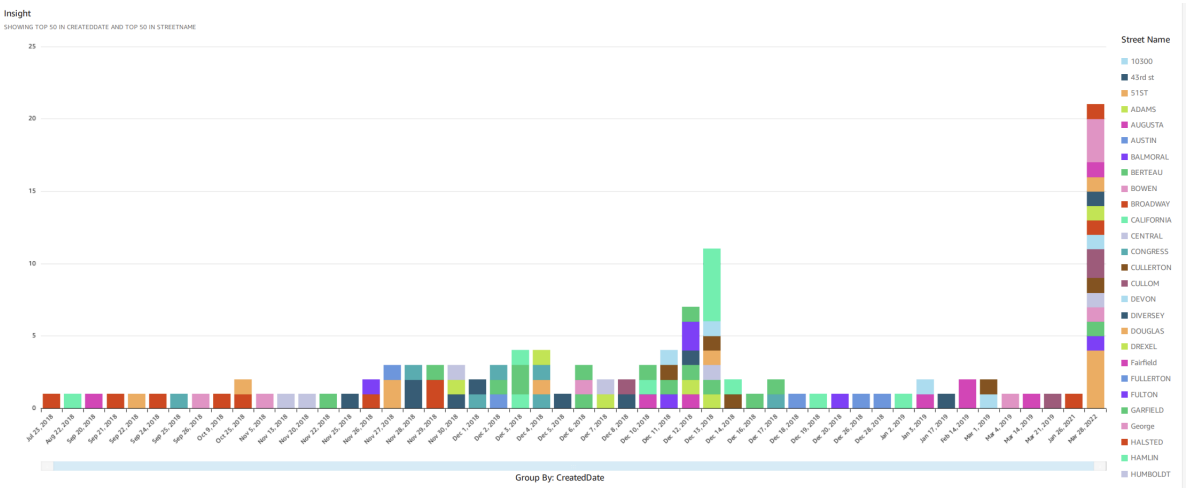


Fig 4.14.3 Insights

Number of service requests raised by the user based on the zipcode.

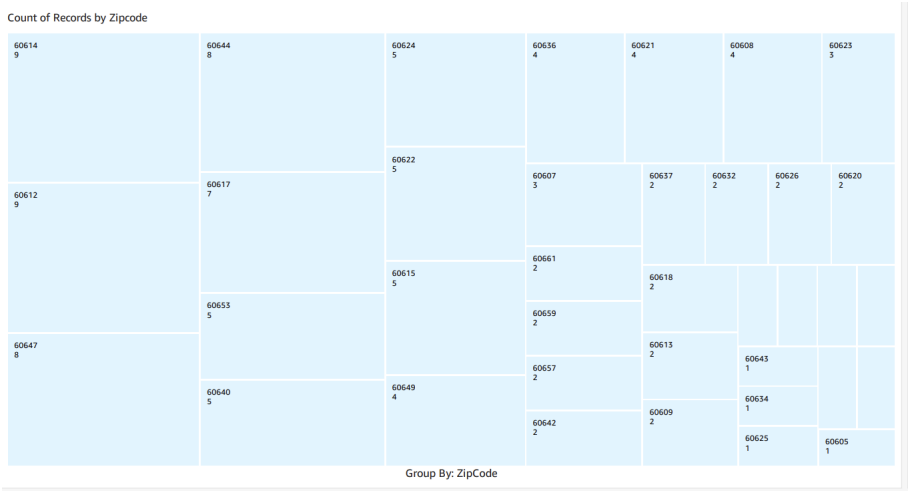


Fig 4.14.4 Count of records by zipcode

Number of service requests raised by the user based on the Service Request Code(SR Code).

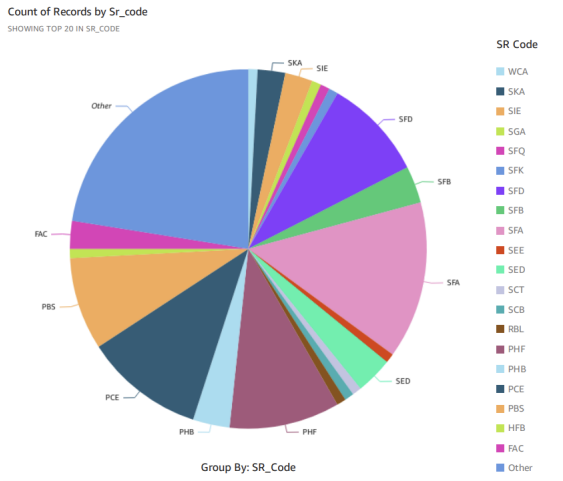


Fig 4.14.5 Group by sr_code

Number of service requests based on the request status (Completed, Canceled and Open)

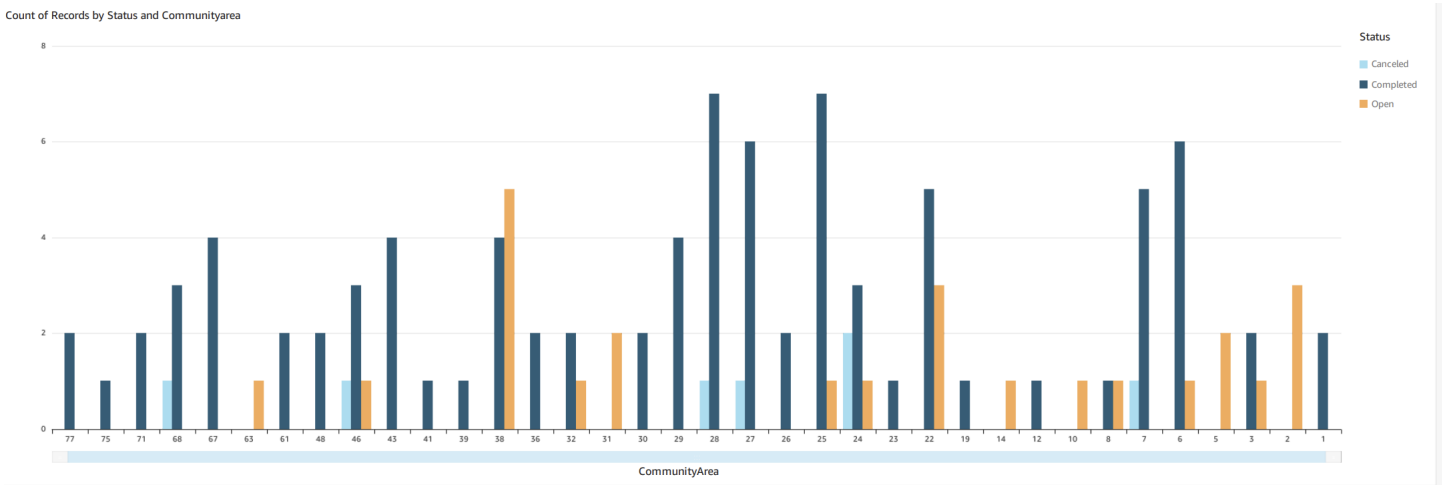


Fig 4.14.6 Group by status and community area

Number of Service requests created by Street Name

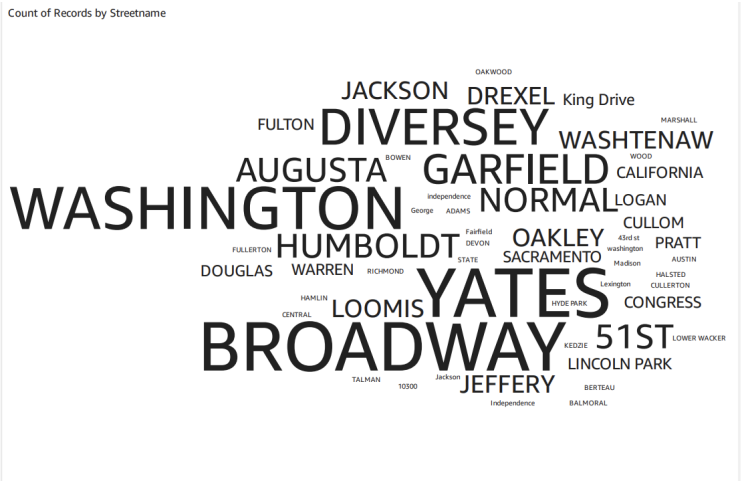


Fig 4.14.7 Group by street name

Service requests created based on the Department

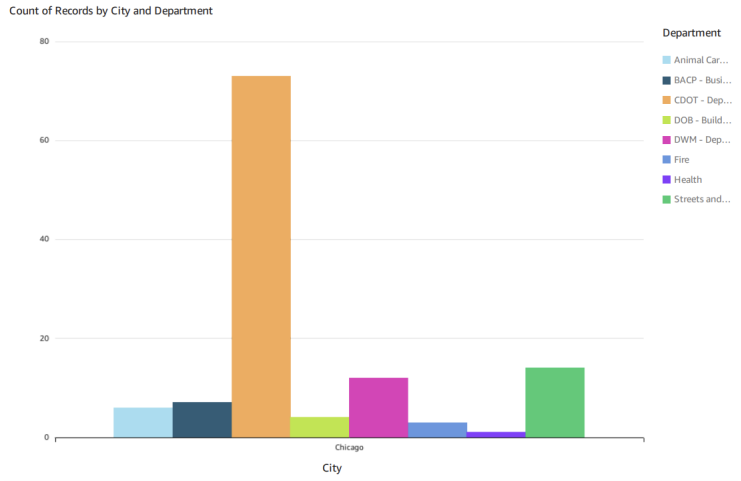


Fig 4.14.8 Group by city and department

Google Map Zipcode view to analyze the number requests raised by users in a ZipCode

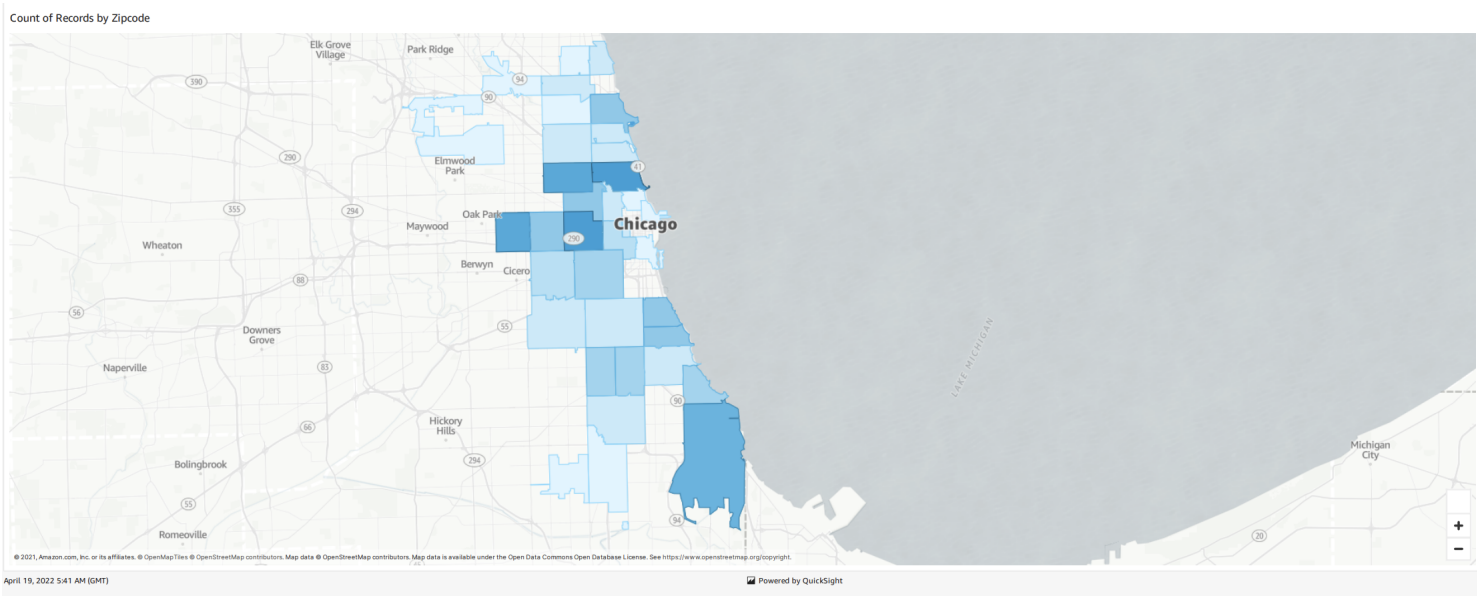


Fig 4.14.9 Group by zipcode

5 DATASET

The dataset used in the pipeline is 311 Service Requests data received by the City of Chicago.[2] This dataset includes requests created after 07/23/2018. This dataset has to be cleaned as it has many null values and data inconsistencies for a few columns.

6 CONCLUSION

To automate the analytical process, a data pipeline is built using a number of AWS services. To accept and analyze data in real-time, we used tools like S3, lambda, AWS Kinesis, AWS DynamoDB, and Athena. A comparison is conducted between DynamoDB and DocumentDB, Kafka and Kinesis Stream to see which has the edge over the others in terms of building the pipeline. Finally, we performed a real-time visual analysis of readings taken using QuickSight.

References

[1] AWS big data pipeline blog https://aws.amazon.com/blogs/big-data

[2] DataSet https://data.cityofchicago.org/Service-Requests/311-Service-Requests

[3] Eric Johnson. Jun 27, 2019. https://aws.amazon.com/blogs/compute/increasing-real-time-stream-processing-performance-with-amazon-kinesis-data-streams-enhanced-fan-out-and-aws-lambda/

[4] Brewer, E. (2000). Towards Robust Distributed System. Symposium on Principles of Distributed Computing (PODC).

[5] Dynamo: Amazon's Highly Available Key-value Store.

[6] Amazon DynamoDB Developer Guide (2012). https://docs.aws.amazon.com/amazondynamodb

[7] Jame, Curtis. (2022). After 10 Years, AWS continues to expand Amazon DynamoDB well beyond scalability. 2022 SP Global Market Intelligence.

[8] Durgeshkashyap (2021).Choosing the Best NoSQL Database — DocumentDB vs DynamoDB vs MongoDB. https://infohubblog.com/documentdb-vs-dynamodb-vs-mongodb.html

[9] Amazon NeptuneDeveloper Guide (2020). https://docs.aws.amazon.com/neptune/latest/userguide/neptune-ug.pdfintro

[10] Alex, Debrie. (2021). Everything you need to know about DynamoDB Partitions.

[11] Difference Between DocumentDB vs DynamoDB. https://www.educba.com/documentdb-vs-dynamodb/

[12] System Properties Comparison Amazon DocumentDB vs. Amazon DynamoDB vs. Amazon Neptune.