University of New Haven
TAGLIATELA COLLEGE OF ENGINEERING
Electrical & Computer Engineering and Computer Science

Real-Time Website Traffic Analytics Using AWS Cloud Services

DSCI- 6007-01
Distributed & Scalable Data Engineering
DSCI 6007
Instructor: Dr. Ardiana Sula
2025-04-28

Team Members:
Hrishabh Mahaju - hmaha4@unh.newhaven.edu
Yaminipoojitha Rayadurgam - yraya1@unh.newhaven.edu
Sahil Ghelani - sghel1@unh.newhaven.edu

GitHub Repository:
https://github.com/hrishabh360/NasaWebsiteLogsDataEngineering/tree/master

CONTENTS

Team Member Role Assignment

Team Member	Role	Responsibilities
Hrishabh Mahaju	Project Manager & Data Engineer	- Oversee the project - Lead architecture design - Set up AWS services (Kinesis, S3, Lambda) - Design and implement the data pipeline
Yaminipoojitha Rayadurgam	Data Ingestion and Processing Engineer	- Develop Python script for log parsing - Set up data streaming to Kinesis - Assist in Lambda function development - Handle schema design for Glue/Athena
Sahil Ghelani	Data Analyst and Visualization Specialist	- Query data using Athena - Design visualizations in Power BI - Analyze traffic patterns, status codes, and user behaviors - Prepare data insights for the report and presentation

Teamwork Reflection

Throughout the project, effective teamwork and collaboration were key to the team's success. Each member brought their strengths to specific stages of the project, simulating a real-world distributed data engineering environment.

As the Team Leader and Project Manager, Hrishabh was responsible for overseeing the entire project lifecycle, coordinating tasks, and ensuring timely execution. Hrishabh also led the AWS architecture setup, configuring services such as Kinesis, Lambda, and S3.

Yaminipoojitha, acting as the Data Ingestion and Processing Engineer, developed the Python scripts to parse and stream NASA HTTP logs into Kinesis and contributed to the Lambda function that processed real-time records.

Sahil, in the role of Data Analyst and Visualization Specialist, focused on querying the processed data using Athena, building insightful visualizations in Power BI, and extracting meaningful traffic patterns to support business understanding.

The team maintained consistent communication through scheduled meetings, shared GitHub repositories, and collaborative planning sessions. Tasks were divided according to each member’s expertise, promoting efficiency and continuous learning. Challenges such as limited AWS account permissions were collectively identified and resolved through brainstorming and research.

The distributed teamwork approach not only ensured the technical success of the project but also simulated professional data engineering collaboration, fostering accountability, technical skill development, and shared ownership of the project's outcomes.

.

Abstract

In the fast-paced realm of system operations and software development, reading website logs is an essential task for monitoring system health, enhancing user experience, and decision-making. Drawing upon past industry experience as a Software Engineer, Hrishabh saw the potential for applying real-time website traffic logs to develop useful insights that influence the Software Development Life Cycle (SDLC). Having seen this, the team conceptualized and deployed a real-time website traffic analytics pipeline using scalable cloud technologies.

The solution is to implement a full end-to-end data engineering pipeline to simulate the ingestion, processing, storage, and analysis of real real-time web server logs. NASA's HTTP Web Server Logs were used as the source dataset to simulate high-speed traffic information. A Python script loads the logs into Amazon Kinesis where AWS Lambda functions to process the records and store them as structured JSON objects in Amazon S3. It is then cataloged by AWS Glue and queried by Amazon Athena, and finally, analysis is visualized by Power BI.

This project demonstrates a distributed, scalable system that can inspect real-time web site traffic, identify the most popular pages, monitor status code distributions, and identify traffic patterns over time. The team, with this pipeline, demonstrates the value of log analysis in contemporary system administration and demonstrates the way real-time observations can influence improved performance monitoring, system debugging, and user experience work. The solution emphasizes cloud-native, serverless architecture aligned with today's data engineering practices.

Introduction

In the digital era, website traffic logs represent a valuable source of information for understanding user behavior, system performance, and potential operational bottlenecks. Organizations rely heavily on real-time insights from server logs to enhance decision-making, optimize user experiences, and proactively address technical issues. Inspired by practical industry experience in software engineering, this project was designed to explore how real-time log analysis can support the Software Development Life Cycle (SDLC) and operational excellence.

The primary objective of this project is to build a scalable, distributed data engineering pipeline capable of ingesting, processing, storing, and analyzing real-time website traffic logs. Using NASA's HTTP Web Server Logs as the foundational dataset, the project simulates high-velocity traffic and demonstrates how modern cloud-native tools can be leveraged to handle real-time data streams efficiently.

Amazon Kinesis serves as the ingestion layer for streaming data, while AWS Lambda functions process and structure the logs. Amazon S3 provides a reliable and scalable storage solution, and AWS Glue facilitates schema inference and cataloging. The processed data is queried using Amazon Athena and visualized through Power BI, enabling interactive analysis of traffic patterns, status codes, and user interactions.

This project reflects the core principles of distributed and scalable data engineering, integrating cloud-based services to create a seamless end-to-end pipeline. It showcases the importance of real-time data analytics and demonstrates how actionable insights can be derived from seemingly raw and unstructured web server logs.

Literature Review

A critical step in understanding the relevance of website traffic log analysis is to review prior research and existing industry practices in the domain of real-time data ingestion, processing, and analytics. The growing demand for real-time system monitoring, fault detection, and user behavior analysis has driven significant advancements in scalable log analytics pipelines.

Previous studies have explored the importance of server log analysis in system operations and cybersecurity. Early frameworks, as described by Lee and Stolfo (2000), utilized batch processing models to retrospectively extract insights from large web server logs. However, the evolution of cloud computing and stream processing technologies has enabled a shift toward real-time analytics, significantly reducing detection and response times (Marz & Warren, 2015).

Amazon Web Services (AWS) technical whitepapers emphasize the use of event-driven, serverless architectures — particularly services like Amazon Kinesis and AWS Lambda — for building highly scalable, fault-tolerant real-time data pipelines (AWS Architecture Center, 2020). These services allow near-instantaneous ingestion and transformation of log records, enabling system administrators to monitor performance, detect anomalies, and influence operational decisions rapidly.

Historical datasets, such as NASA's HTTP Web Server Logs, have been widely used in academic settings to simulate high-traffic scenarios and benchmark real-time data systems. Studies such as Zaharia et al. (2012) demonstrated the effectiveness of distributed systems like Spark Streaming for managing fast data streams, influencing modern cloud-native designs that favor managed services over self-hosted clusters.

Furthermore, the integration of tools such as AWS Glue and Amazon Athena facilitates seamless cataloging and querying of structured data derived from unstructured logs. Visualization platforms like Power BI have also been recognized for translating complex analytics outputs into actionable insights for both technical and non-technical stakeholders (Chaudhuri, Dayal, & Narasayya, 2011).

Overall, the literature highlights the growing importance of real-time web log analysis in the Software Development Life Cycle (SDLC), particularly for proactive debugging, performance optimization, and enhancing user experiences. Building upon these foundations, the current project implements a distributed, serverless pipeline aligned with contemporary best practices to demonstrate the critical role of cloud-native log analytics in modern system administration and development workflows.

Methodology

This project employs the CRISP-DM (Cross Industry Standard Process for Data Mining) methodology to systematically organize the data engineering, processing, and analysis tasks.

The project, titled "Real-Time Website Traffic Analytics Using Distributed and Scalable AWS Cloud Services," focuses on demonstrating how real-time website traffic logs can be leveraged to generate meaningful insights that aid operational decision-making and enhance system performance.

Business Understanding:
Recognizing the importance of log analysis in the Software Development Life Cycle (SDLC), the project aims to showcase how real-time analytics on website traffic can improve system monitoring, detect anomalies, and support infrastructure optimization.

Data Understanding:
The NASA HTTP Web Server Logs from July 1995 were chosen due to their real-world structure and high-frequency request patterns. These logs contain crucial attributes such as IP addresses, timestamps, HTTP methods, URLs, status codes, and response sizes.

Data Preparation:
A Python-based ingestion script was developed to parse and convert the semi-structured logs into structured JSON format. Data was streamed into Amazon Kinesis, processed by AWS Lambda functions, and stored in Amazon S3. AWS Glue was used to catalog the data, preparing it for query and analysis.

Modeling:
The pipeline architecture utilized Amazon Kinesis for real-time ingestion, AWS Lambda for on-the-fly processing, Amazon S3 for storage, AWS Glue for schema detection, Athena for querying, and Power BI for visualization. This serverless, scalable model closely mirrors modern industry best practices for real-time data engineering.

Evaluation:
The system was evaluated based on the successful ingestion, processing, and storage of log data, and the ability to derive insights through Athena queries and Power BI dashboards. Visualizations such as top URLs, traffic timelines, and HTTP status code distributions validated the effectiveness of the pipeline in extracting operational intelligence from raw log data.
This methodology ensures a comprehensive, scalable, and practical approach to real-time website traffic analytics using cloud-native technologies.

Project Workflow Diagram

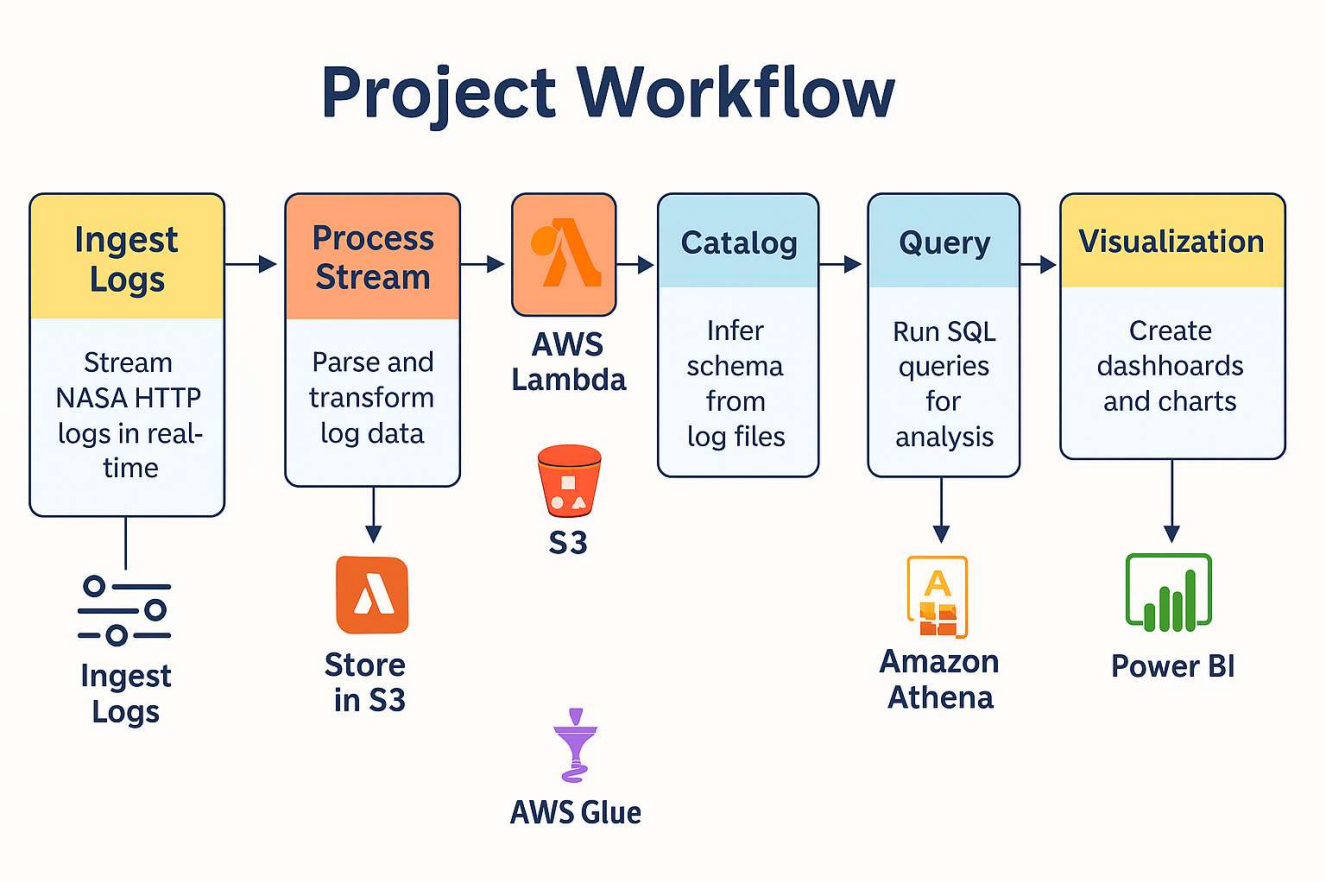


Fig. Project workflow diagram for Real-Time Website Traffic Analytics Using AWS Cloud Services

The project workflow begins with the ingestion of NASA HTTP server logs, which are streamed in real-time using a Python script into Amazon Kinesis. From there, AWS Lambda functions process and transform the incoming data into a structured JSON format. The processed data is then stored in Amazon S3, providing scalable and durable storage. AWS Glue is used to catalog the stored data and infer the schema, making it query able. Using Amazon Athena, SQL queries are executed on the structured data to extract meaningful insights. Finally, the results are visualized through Power BI dashboards and charts, enabling interactive analysis of traffic patterns, popular URLs, and system health metrics.

Findings

The Power BI dashboard developed for this project provides a comprehensive overview of real-time website traffic behavior derived from the NASA HTTP Web Server Logs. Key insights from the visualizations are summarized below:

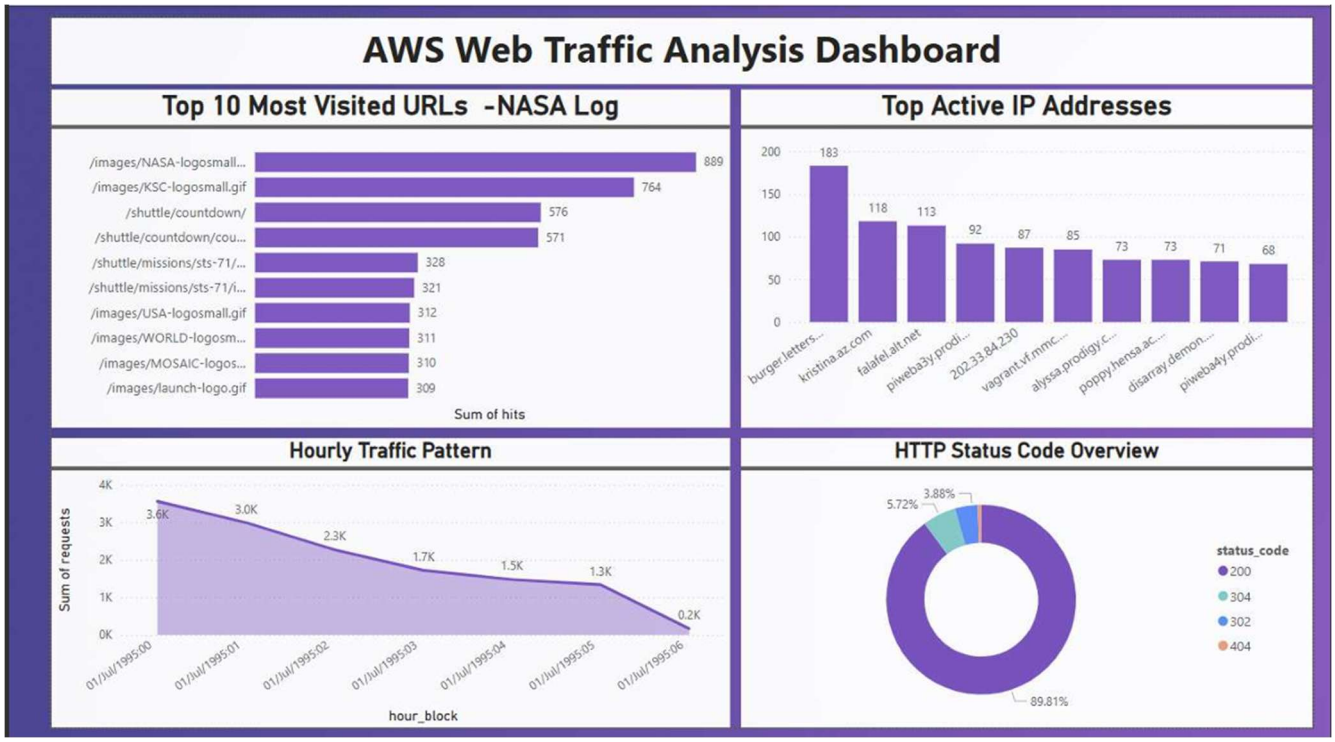


Fig. NASA HTTP Web Server Logs Power-BI Dashboard.

- **1. Top 10 Most Visited URLs**

SQL Query:

```
-- Most Requested URLs

SELECT url

,COUNT(*) AS hits

FROM nasa_logs_nasa_kinesis_output_logs

GROUP BY url
```

```
ORDER BY hits DESC LIMIT 10;
```

The most frequently accessed resources on the server were primarily image files, such as /images/NASA-logosmall.gif and /images/KSC-logosmall.gif, each with hundreds of requests. This suggests that users heavily accessed graphical content, possibly for educational or media usage. Pages related to the /shuttle/countdown/ and mission logs also ranked high, indicating significant user interest in specific NASA missions and launch events.

- **2. Top Active IP Addresses**

SQL Query:

```
-- Top IP Addresses (most active users)

SELECT ip

   ,COUNT(*) AS request_count

FROM nasa_logs_nasa_kinesis_output_logs

GROUP BY ip

ORDER BY request_count DESC LIMIT 10;
```

Analysis of IP activity revealed a handful of highly active IP addresses, with the top contributor (burger.letters.com) generating nearly 200 requests. This pattern may reflect institutional scraping, automated data retrieval, or high user engagement from specific networks or locations.

- **3. Hourly Traffic Pattern**

SQL Query:

```
-- Traffic Over Time (Hourly Breakdown)

SELECT substr(TIMESTAMP, 1, 14) AS hour_block

   ,COUNT(*) AS requests

FROM nasa_logs_nasa_kinesis_output_logs

GROUP BY substr(TIMESTAMP, 1, 14)

ORDER BY hour_block;
```

The line chart of hourly traffic reveals peak activity between midnight and early morning hours (specifically between 00:00 and 02:00 UTC), with request volume gradually tapering off throughout the

day. This trend may be attributed to global interest across different time zones or automated traffic concentrated in early hours.

4. HTTP Status Code Overview

SQL Query:

```
-- HTTP Status Code Distribution

SELECT status_code

   ,COUNT(*) AS count

FROM nasa_logs_nasa_kinesis_output_logs

GROUP BY status_code

ORDER BY count DESC;
```

Most requests (~90%) returned a 200 OK status, confirming that most resources were successfully served. A small percentage of 302 and 304 responses represent redirections and cache validations, respectively. The presence of 404 Not Found errors (~1.5%) also highlights some broken or outdated links within the logs.

Interpretation

These findings demonstrate the practical value of log data analysis in understanding user access patterns, identifying popular content, monitoring system health, and detecting anomalies. By analyzing traffic trends, frequent request sources, and response codes, organizations can make data-driven improvements to performance, usability, and content delivery.

Discussion

The findings from this project affirm the practical relevance of real-time web log analysis in system monitoring, traffic optimization, and user behavior tracking. From the outset, our research question centered around understanding whether serverless, distributed architecture could be used to transform unstructured log data into actionable insights for system-level decision-making. The results show that this approach is both feasible and highly effective.

The visualizations generated through Power BI clearly demonstrate patterns in user access behavior. For instance, the overwhelming interest in graphical resources like NASA mission logos points to a content-driven traffic structure, potentially from media sites or academic usage. This validates the assumption that logs are not merely technical artifacts but reflections of user intent.

Equally important are the patterns observed in IP activity and hourly usage. Identifying high-volume IPs or unusual surges in traffic equips system administrators with the tools to flag scraping behavior, detect bot traffic, or plan for server scaling. The hourly traffic decline, for example, reveals time-zone-driven engagement, further showcasing how operational strategies can be shaped through log-derived evidence.

However, as with all analytical processes, the findings offer a partial view. This dataset reflects a single month from 1995, meaning broader trends across months or modern systems are not represented. Additionally, since the NASA logs were anonymized and cleaned, the project assumes generalizability without incorporating elements like user sessions, geolocation, or authentication logs.

Still, this project provides a strong proof-of-concept. It bridges a clear knowledge gap between real-time data engineering and business analytics. By successfully deploying and evaluating a fully cloud-native log analysis pipeline, the project reinforces the value of integrating scalable tools like AWS and Power BI into modern SDLC practices — not only for developers but for analysts, system architects, and decision-makers alike.

Conclusion

This project successfully demonstrated the design and implementation of a real-time, serverless data pipeline to analyze website traffic logs using modern cloud technologies. By simulating real-world traffic through NASA's HTTP Web Server Logs and applying a distributed architecture involving Amazon Kinesis, Lambda, S3, Glue, and Athena, the project showcased how scalable systems can transform raw log data into valuable operational insights.

Through structured data processing and cloud-native orchestration, the pipeline enabled near real-time analysis of user access patterns, top URLs, HTTP status distributions, and traffic trends. The final visualizations created in Power BI illustrated how technical data can be transformed into intuitive dashboards that inform system monitoring, debugging, and content strategy.

The project also highlights the value of adopting cloud services in data engineering education and practice. It emphasized not only technical implementation but also teamwork, problem-solving, and architectural thinking in a simulated industry environment. The use of serverless components ensured low overhead and high scalability — aligning with best practices in modern data infrastructure.

While the dataset and scope were constrained for academic purposes, the methodology is extensible and adaptable to live systems. Future enhancements could include incorporating more recent log sources, applying user segmentation, performing geolocation analysis, or integrating machine learning models for anomaly detection.

Overall, this project reinforces the critical role of real-time log analytics in supporting system resilience, user-centric development, and operational intelligence — key pillars of today’s data-driven organizations.

Contributions/References

1. Lee, W., & Stolfo, S. J. (2000). A framework for constructing features and models for intrusion detection systems.
2. Marz, N., & Warren, J. (2015). Big Data: Principles and best practices of scalable real-time data systems.
3. AWS Architecture Center (2020). Serverless Architectures with AWS Lambda: Overview and Best Practices.
4. Zaharia, M., Das, T., Li, H., Hunter, T., Shenker, S., & Stoica, I. (2012). Discretized Streams: Fault-tolerant streaming computation at scale.
5. Chaudhuri, S., Dayal, U., & Narasayya, V. (2011). An overview of business intelligence technology.

Appendices:

parser.py > ...

```python
import re
import json

log_file = "access_log_Jul95"

# Regex pattern to parse the log line
log_pattern = re.compile(
    r'(?P<ip>\S+) - - \[(?P<timestamp>[^\]]+)\] '
    r'"(?P<method>\S+)\s(?P<url>\S+)\s(?P<protocol>[^"]+)" '
    r'(?P<status_code>\d{3}) (?P<bytes>\S+)'
)

# Read and parse log lines
with open(log_file, 'r') as f:
    for i, line in enumerate(f):
        match = log_pattern.match(line)
        if match:
            log_json = match.groupdict()
            # Handle '-' in bytes as 0
            log_json['bytes'] = int(log_json['bytes']) if log_json['bytes'].isdigit() else 0
            log_json['status_code'] = int(log_json['status_code'])
            print(json.dumps(log_json))
        if i == 10:  # Print only first 10 lines for now
            break

```

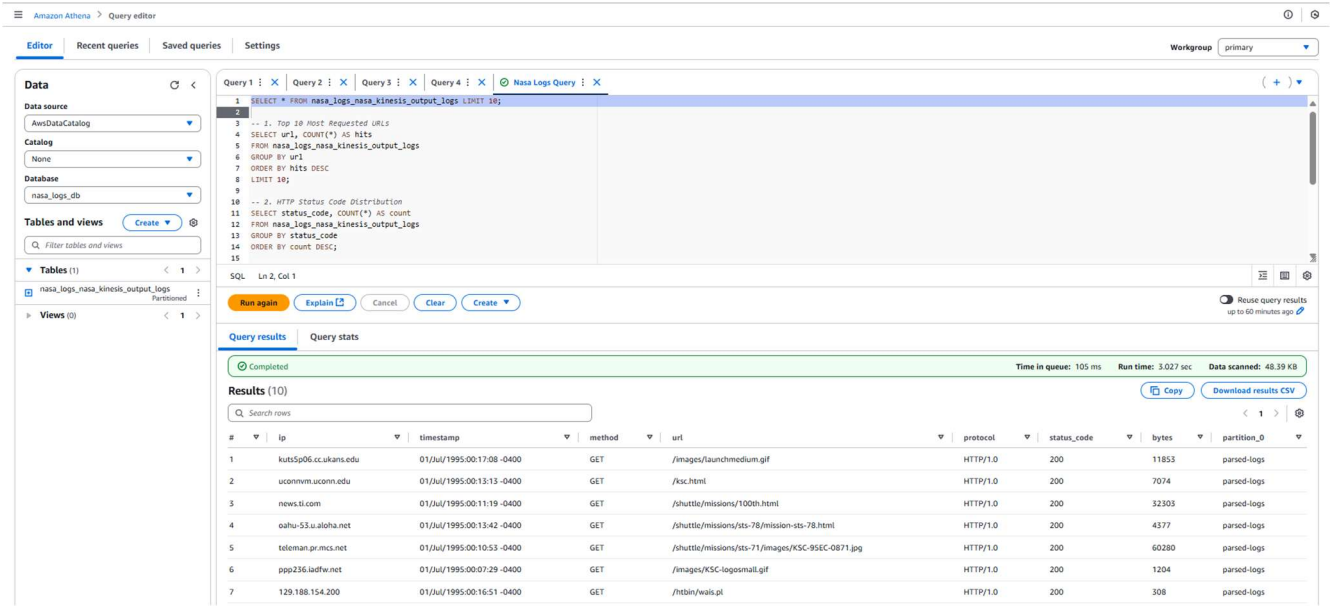
lambda_kinesis_to_s3.py > ...

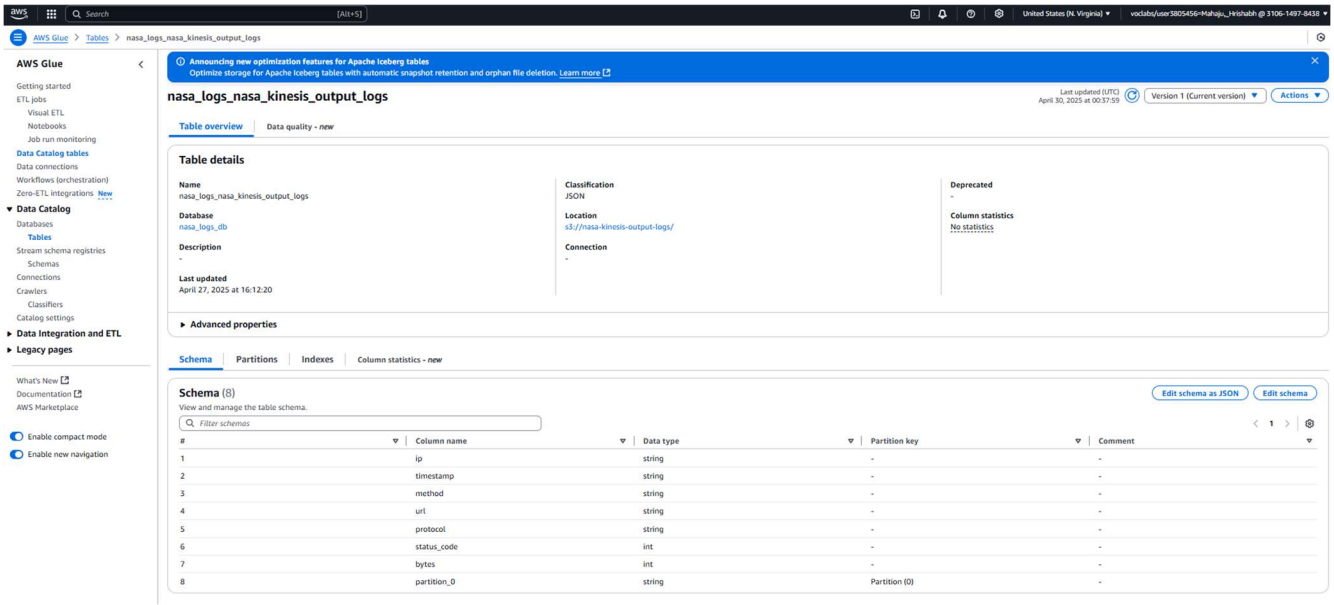
```python
import json
import boto3
import base64
import uuid
from datetime import datetime

s3 = boto3.client('s3')
bucket_name = 'nasa-kinesis-output-logs'  # ✅ Confirm this is exactly your bucket name

def lambda_handler(event, context):
    for record in event['Records']:
        payload = base64.b64decode(record['kinesis']['data']).decode('utf-8')
        print("Raw Payload:", payload)  # ✅ Log raw input

        try:
            data = json.loads(payload)
            print("Parsed JSON:", data)
        except json.JSONDecodeError as e:
            print("JSON Decode Error:", e)
            continue

        log_key = f"parsed-logs/{datetime.utcnow().strftime('%Y-%m-%d_%H-%M-%S')}_{uuid.uuid4()}.json"
        print(f"Writing to S3 → Bucket: {bucket_name}, Key: {log_key}")

        try:
            s3.put_object(
                Bucket=bucket_name,
                Key=log_key,
                Body=json.dumps(data),
                ContentType='application/json'
            )
            print("✅ Upload successful")
        except Exception as e:
            print("❌ S3 upload failed:", e)

    return {
        'statusCode': 200,
        'body': json.dumps('Finished processing records.')
    }
```

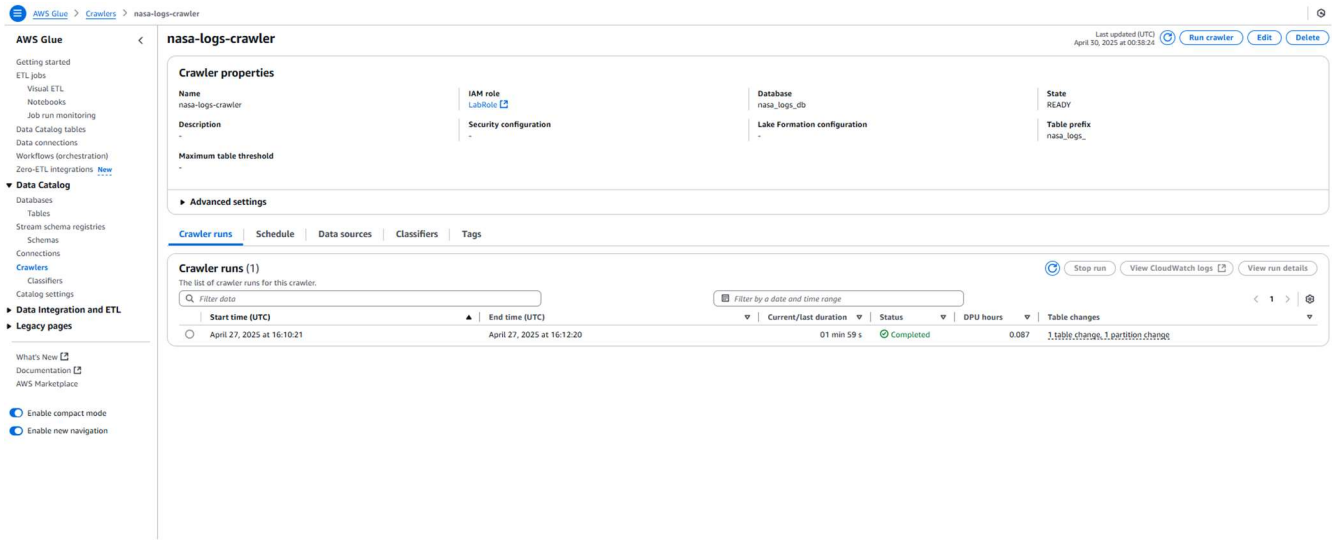
README.md | send_to_kinesis.py

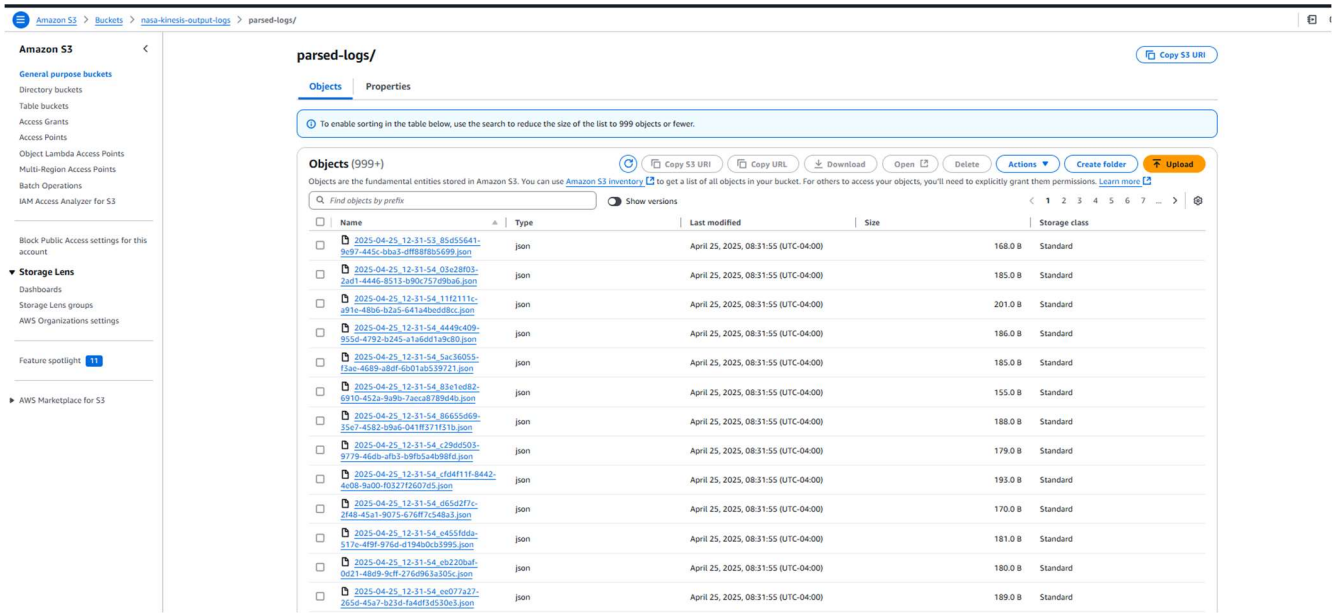
send_to_kinesis.py > ...

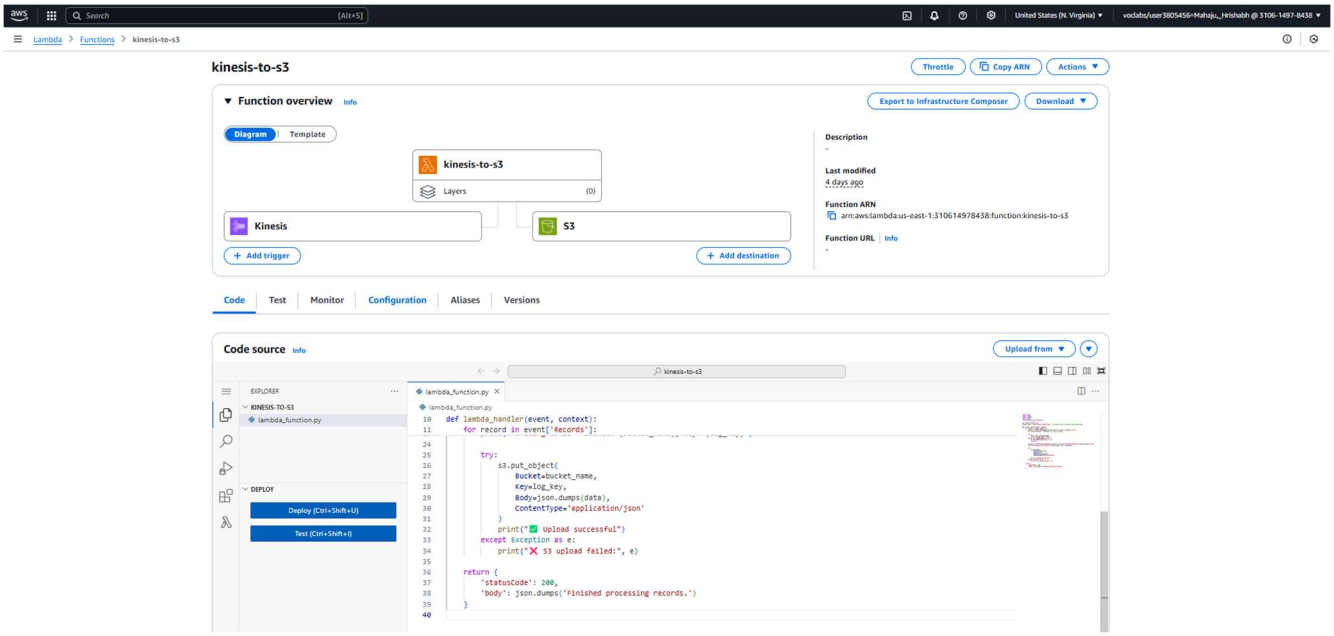
```python
import boto3
import json
import re
import time
import random

# Load AWS credentials from local JSON file
with open("aws_credentials.json") as cred_file:
    creds = json.load(cred_file)

# Config
LOG_FILE = r"access_log_Jul95"  # 🔁 Replace with your actual file path
STREAM_NAME = "website-traffic-stream"              # 🔁 Make sure this matches your Kinesis stream
REGION = creds["region"]

# Create Kinesis client using manual credentials
kinesis = boto3.client(
    "kinesis",
    region_name=REGION,
    aws_access_key_id=creds["aws_access_key_id"],
    aws_secret_access_key=creds["aws_secret_access_key"],
    aws_session_token=creds["aws_session_token"]
)

# Log line regex pattern
pattern = re.compile(
    r'(?P<ip>\S+) - - \[(?P<timestamp>[^\]]+)\] '
    r'"(?P<method>\S+)\s(?P<url>\S+)\s(?P<protocol>[^"]+)" '
    r'(?P<status_code>\d{3}) (?P<bytes>\S+)'
)

# Stream log entries to Kinesis
with open(LOG_FILE, "r") as f:
    for line in f:
        match = pattern.match(line)
        if match:
            record = match.groupdict()
            record["bytes"] = int(record["bytes"]) if record["bytes"].isdigit() else 0
            record["status_code"] = int(record["status_code"])

            # Send to Kinesis
            kinesis.put_record(
                StreamName=STREAM_NAME,
                Data=json.dumps(record),
```


Amazon Athena > Query editor
Editor
Recent queries
Saved queries
Settings
Workgroup primary
Data
Data source
AwsDataCatalog
Catalog
None
Database
nasa_logs_db
Tables and views
Create
Filter tables and views
Tables (1)
nasa_logs_nasa_kinesis_output_logs
Partitioned
Views (0)
Query 1
Query 2
Query 3
Query 4
Nasa Logs Query
SELECT * FROM nasa_logs_nasa_kinesis_output_logs LIMIT 10;
-- 1. Top 10 Most Requested URLs
SELECT url, COUNT(*) AS hits
FROM nasa_logs_nasa_kinesis_output_logs
GROUP BY url
ORDER BY hits DESC
LIMIT 10;
-- 2. HTTP Status Code Distribution
SELECT status_code, COUNT(*) AS count
FROM nasa_logs_nasa_kinesis_output_logs
GROUP BY status_code
ORDER BY count DESC;
SQL Ln 2, Col 1
Run again
Explain
Cancel
Clear
Create
Reuse query results
up to 60 minutes ago
Query results
Query stats
Completed
Time in queue: 105 ms
Run time: 3.027 sec
Data scanned: 48.39 KB
Results (10)
Copy
Download results CSV
Search rows
ip timestamp method url protocol status_code bytes partition_0
1 kuts5p06.cc.ukans.edu 01/Jul/1995:00:17:08 -0400 GET /images/launchmedium.gif HTTP/1.0 200 11853 parsed-logs
2 uconnvm.uconn.edu 01/Jul/1995:00:13:13 -0400 GET /ksc.html HTTP/1.0 200 7074 parsed-logs
3 news.ti.com 01/Jul/1995:00:11:19 -0400 GET /shuttle/missions/100th.html HTTP/1.0 200 32303 parsed-logs
4 oahu-53.u.aloha.net 01/Jul/1995:00:13:42 -0400 GET /shuttle/missions/sts-78/mission-sts-78.html HTTP/1.0 200 4377 parsed-logs
5 teleman.pr.mcs.net 01/Jul/1995:00:10:53 -0400 GET /shuttle/missions/sts-71/images/KSC-95EC-0871.jpg HTTP/1.0 200 60280 parsed-logs
6 ppp236.iadfw.net 01/Jul/1995:00:07:29 -0400 GET /images/KSC-logosmall.gif HTTP/1.0 200 1204 parsed-logs
7 129.188.154.200 01/Jul/1995:00:16:51 -0400 GET /htbin/wais.pl HTTP/1.0 200 308 parsed-logs


AWS Glue > Tables > nasa_logs_nasa_kinesis_output_logs
AWS Glue
Getting started
ETL jobs
Visual ETL
Notebooks
Job run monitoring
Data Catalog tables
Data connections
Workflows (orchestration)
Zero-ETL integrations New
Data Catalog
Databases
Tables
Stream schema registries
Schemas
Connections
Crawlers
Classifiers
Catalog settings
Data Integration and ETL
Legacy pages
What's New
Documentation
AWS Marketplace
Enable compact mode
Enable new navigation
Announcing new optimization features for Apache Iceberg tables
Optimize storage for Apache Iceberg tables with automatic snapshot retention and orphan file deletion. Learn more
nasa_logs_nasa_kinesis_output_logs
Last updated (UTC) April 30, 2025 at 00:37:59
Version 1 (Current version)
Actions
Table overview
Data quality - new
Table details
Name
nasa_logs_nasa_kinesis_output_logs
Database
nasa_logs_db
Description
-
Last updated
April 27, 2025 at 16:12:20
Classification
JSON
Location
s3://nasa-kinesis-output-logs/
Connection
-
Deprecated
-
Column statistics
No statistics
Advanced properties
Schema
Partitions
Indexes
Column statistics - new
Schema (8)
View and manage the table schema.
Edit schema as JSON
Edit schema
Filter schemas
Column name Data type Partition key Comment
1 ip string - -
2 timestamp string - -
3 method string - -
4 url string - -
5 protocol string - -
6 status_code int - -
7 bytes int - -
8 partition_0 string Partition (0) -


AWS Glue > Crawlers > nasa-logs-crawler
AWS Glue
Getting started
ETL jobs
Visual ETL
Notebooks
Job run monitoring
Data Catalog tables
Data connections
Workflows (orchestration)
Zero-ETL integrations New
Data Catalog
Databases
Tables
Stream schema registries
Schemas
Connections
Crawlers
Classifiers
Catalog settings
Data Integration and ETL
Legacy pages
What's New
Documentation
AWS Marketplace
Enable compact mode
Enable new navigation
nasa-logs-crawler
Last updated (UTC) April 30, 2025 at 00:38:24
Run crawler
Edit
Delete
Crawler properties
Name
nasa-logs-crawler
IAM role
LabRole
Database
nasa_logs_db
State
READY
Description
-
Security configuration
-
Lake Formation configuration
-
Table prefix
nasa_logs_
Maximum table threshold
-
Advanced settings
Crawler runs
Schedule
Data sources
Classifiers
Tags
Crawler runs (1)
The list of crawler runs for this crawler.
Stop run
View CloudWatch logs
View run details
Filter data
Filter by a date and time range
Start time (UTC)
End time (UTC)
Current/last duration
Status
DPU hours
Table changes
April 27, 2025 at 16:10:21
April 27, 2025 at 16:12:20
01 min 59 s
Completed
0.087
1 table change, 1 partition change


Amazon S3 > Buckets > nasa-kinesis-output-logs > parsed-logs/
Amazon S3
General purpose buckets
Directory buckets
Table buckets
Access Grants
Access Points
Object Lambda Access Points
Multi-Region Access Points
Batch Operations
IAM Access Analyzer for S3
Block Public Access settings for this account
Storage Lens
Dashboards
Storage Lens groups
AWS Organizations settings
Feature spotlight 11
AWS Marketplace for S3
parsed-logs/
Copy S3 URI
Objects
Properties
To enable sorting in the table below, use the search to reduce the size of the list to 999 objects or fewer.
Objects (999+)
Copy S3 URI
Copy URL
Download
Open
Delete
Actions
Create folder
Upload
Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more
Find objects by prefix
Show versions
Name
Type
Last modified
Size
Storage class
2025-04-25_12-31-53_85d55641-9e97-445c-bba3-dff88f8b5699.json
2025-04-25_12-31-54_03e28f03-2ad1-4446-8513-b90c757d9ba6.json
2025-04-25_12-31-54_11f2111c-a91e-48b6-b2a5-641a4bedd8cc.json
2025-04-25_12-31-54_4449c409-955d-4792-b245-a1a6dd1a9c80.json
2025-04-25_12-31-54_5ac36055-f3ae-4689-a8df-6b01ab539721.json
2025-04-25_12-31-54_83e1ed82-6910-452a-9a9b-7acca8789d4b.json
2025-04-25_12-31-54_86655d69-35e7-4582-b9a6-041ff371f31b.json
2025-04-25_12-31-54_c29dd503-9779-46db-afb3-b9fb5a4b98fd.json
2025-04-25_12-31-54_cfd4f11f-8442-4c08-9a00-f0327f2607d5.json
2025-04-25_12-31-54_d65d2f7c-2f48-45a1-9075-676ff7c548a3.json
2025-04-25_12-31-54_e455fdda-517e-4f9f-976d-d194b0cb3995.json
2025-04-25_12-31-54_eb220baf-0d21-48d9-9cff-276d963a305c.json
2025-04-25_12-31-54_ee077a27-265d-45a7-b23d-fa4df3d530e3.json
json
April 25, 2025, 08:31:55 (UTC-04:00)
168.0 B
185.0 B
201.0 B
186.0 B
185.0 B
155.0 B
188.0 B
179.0 B
193.0 B
170.0 B
181.0 B
180.0 B
189.0 B
Standard


Search
[Alt+S]
United States (N. Virginia)
Lambda > Functions > kinesis-to-s3
kinesis-to-s3
Throttle
Copy ARN
Actions
Function overview Info
Export to Infrastructure Composer
Download
Diagram
Template
kinesis-to-s3
Layers
(0)
Kinesis
S3
Add trigger
Add destination
Description
Last modified
4 days ago
Function ARN
arn:aws:lambda:us-east-1:310614978438:function:kinesis-to-s3
Function URL Info
Code
Test
Monitor
Configuration
Aliases
Versions
Code source Info
Upload from
EXPLORER
KINESIS-TO-S3
lambda_function.py
DEPLOY
Deploy (Ctrl+Shift+U)
Test (Ctrl+Shift+I)
def lambda_handler(event, context):
for record in event['Records']:
try:
s3.put_object(
Bucket=bucket_name,
Key=log_key,
Body=json.dumps(data),
ContentType='application/json'
)
print("✅ Upload successful")
except Exception as e:
print("❌ S3 upload failed:", e)
return {
'statusCode': 200,
'body': json.dumps('Finished processing records.')
}