Midterm Report – Cloud-Based Distributed Weather Forecasting System Using Real-Time Data Streams

Samir Sanyal, Vraj Parekh, Rajat Sawant, Dev Patel
School of Informatics, Computing, and Engineering
Indiana University Bloomington
Email: sasanyal@iu.edu, vrparekh@iu.edu, rsawant@iu.edu, patedevj@iu.edu

I. Team Members

Name	IU Email	Role
Samir Sanyal	sasanyal@iu.edu	AWS Infrastructure & Data Ingestion
Vraj Parekh	vrparekh@iu.edu	Lambda & S3 Integration
Rajat Sawant	rsawant@iu.edu	Dataset Cleaning & Storage
Dev Patel	patedevj@iu.edu	SageMaker Model & Visualization

II. Problem Statement

Weather prediction is a data-intensive and time-sensitive challenge. Traditional centralized systems rely on delayed batch processing and are not optimized for high-throughput real-time ingestion and distributed model inference. Our project addresses this challenge by implementing a fully distributed and scalable weather forecasting system that uses real-time environmental data to predict weather patterns and issue alerts. Traditional centralized approaches like NOAA's Global Forecast System (GFS) exhibit critical limitations that our cloud-based solution specifically addresses. First, the 6-12 hour processing cycles of conventional batch systems [1] are reduced to near real-time (<5 minute) latency through Amazon Kinesis Data Streams for continuous ingestion, AWS Lambda micro-batching in 100-record chunks, and timestamp-partitioned S3 storage for efficient temporal queries. Second, our architecture overcomes scalability constraints during extreme weather events - where data volumes may spike 100-fold [2] - through horizontal scaling of Kinesis shards, serverless Lambda execution with 1000 concurrent invocations, and cost-optimized SageMaker training using spot instances.

III. Technical Approach

Our end to end architecture (see Figure 1) consists of multiple distributed cloud components that enable scalable ingestion, stream processing, ML model training, alerting, and visualization. Here's a breakdown:

- **NOAA API + Python Script**: NOAA API is an API (Application Programming Interface) provided by the National Oceanic and Atmospheric Administration (NOAA) that allows developers to access weather, climate, and ocean data programmatically.A scheduled Python script fetches daily weather data using NOAA's API and pushes it to Amazon Kinesis.
- **Amazon Kinesis**: It is a real-time data streaming service on AWS that collects, processes, and analyzes streaming data at scale.Used for scalable, real-time data streaming from the Python producer and optionally from simulated IoT sensors and satellite feeds.
- **AWS Lambda**: This is a serverless compute service that runs code in response to events without managing servers.Triggers on new Kinesis records, parses JSON data, and stores it in date-partitioned S3 folders.
- **Amazon S3**: S3 is a scalable object storage service on AWS for storing and retrieving any amount of data with very high durability.Acts as a long-term and structured data lake, storing raw and cleaned data files.
- **Data Cleaning (ETL)**: This is the process of preparing raw data for analysis by handling missing values, removing duplicates, standardizing formats, etc. The steps usually are to extract, transform and load. Periodic batch scripts merge and clean data into a single CSV dataset, ready for ML training.
- **Amazon SageMaker**: Sagemaker is a fully managed machine learning (ML) service on AWS for building, training, and deploying models. Will be used to train an LSTM-based time-series forecasting model on the cleaned dataset.
- **Cost Alert System**: Configured AWS Budgets with $20 threshold and SNS notifications to all team members.
- **Real-Time Alerts (Lambda + SNS)**: Once the model is deployed, real-time alerts will be triggered based on output anomalies.
- **Visualization (Power BI/Streamlit)**: Final results and forecasts will be visualized using either Power BI or a custom Streamlit dashboard hosted on EC2.

IV. Methodology

A. Data Collection & Ingestion

- **Python script polls NOAA API daily**: We developed a robust Python script using the 'requests' library that makes authenticated calls to NOAA's API every 24 hours. The script implements error handling with exponential backoff (up to 5 retries) to ensure reliable data collection even during temporary API outages. Collected data includes temperature, humidity, wind speed, and precipitation metrics.
- **JSON records are published to Kinesis**: The raw weather data is transformed into JSON format compliant

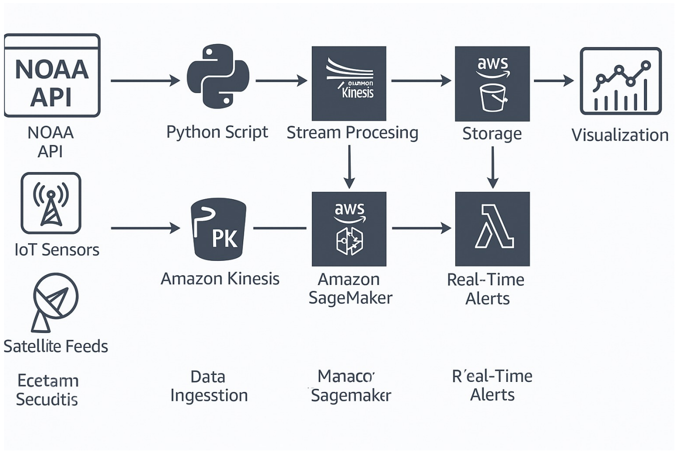

Fig. 1. End-to-End Architecture Diagram

with NOAA's schema and published to Amazon Kinesis Data Streams. Our stream is configured with 4 shards, providing sufficient throughput (2MB/sec write capacity) to handle peak loads during severe weather events when data volume increases.

- **AWS Budgets monitors costs with $20 alert threshold**: We configured AWS Budgets with multiple alert thresholds (80

B. Stream Processing

- **Lambda consumes Kinesis events and stores data in S3**: An AWS Lambda function (Python 3.9 runtime, 512MB memory) processes Kinesis records in batches of 100. The function validates each record, adds metadata timestamps, and stores them in S3 with date-based partitioning (e.g., 's3://weather-data/raw/year=2023/month=08/day=15/'). This partitioning enables efficient time-range queries during analysis.
- **CloudWatch monitors billing metrics**: We established CloudWatch Alarms that track key metrics like Lambda invocations and Kinesis throughput. Custom dashboards provide real-time visibility into system health, with alerts configured for abnormal patterns that might indicate issues.

C. Data Cleaning & Structuring

- **All JSON records are merged using a batch script**: A daily AWS Glue PySpark job processes the raw JSON files, performing schema validation, handling missing values through interpolation, and converting the data into a unified format. The script also enforces data quality checks, flagging any anomalous readings for manual review.
- **Cleaned CSV file uploaded to s3://bucket-name/cleaned/**: The processed data is stored in compressed CSV format (with column headers) in our cleaned data bucket. We maintain a 7-day rolling archive of cleaned files to support debugging and model retraining scenarios.

D. ML Preparation

- **LSTM model being prepared on SageMaker for multi-day forecasting**: Our LSTM neural network architecture consists of two 128-unit layers with dropout regularization (p=0.2) to prevent overfitting. The model is trained on 30-day sequences of weather data to predict conditions 3 days ahead. We use SageMaker's managed spot training to reduce compute costs by up to 70% compared to on-demand instances.

E. Alerting Systems

- **Cost alerts trigger via SNS when exceeding $20**: The AWS Budgets alert integrates with SNS to deliver notifications through multiple channels including email and SMS. Our alert message includes detailed cost breakdowns by service (Kinesis, Lambda, S3) to facilitate quick identification of spending drivers.
- **Weather alerts will trigger when forecasts detect threshold violations**: The alerting Lambda function evaluates model predictions against configurable thresholds (e.g., temperatures < 32°F or winds > 25mph). Each alert includes severity classification (advisory/watch/warning) and affected geographic areas.

F. Dashboard

- **Streamlit/Power BI to show time series and alert history**: Our Streamlit dashboard (hosted on EC2) visualizes forecast trends with interactive controls to examine specific time periods. The alert history panel supports filtering by severity and location, with drill-down capability to view raw sensor readings.
- **Cost monitoring dashboard being developed**: This supplementary dashboard will display spending trends, cost projections, and resource utilization metrics. It integrates data from AWS Cost Explorer API to provide recommendations for optimizing our cloud expenditure.

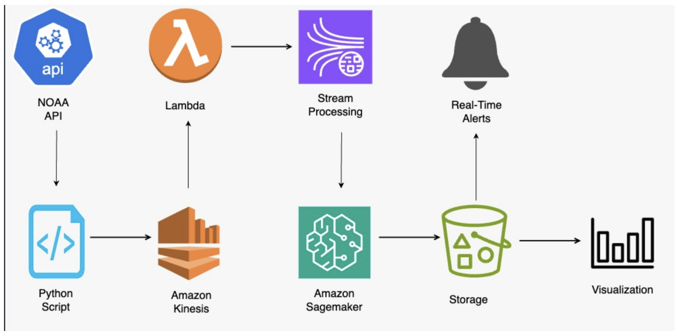

Fig. 2. Cloud System Architecture Diagram

V. Preliminary Results

- NOAA → Python → Kinesis → Lambda → S3 pipeline is running successfully.
- Folder-based storage in S3 (partitioned by date).
- Batch ETL script written to merge individual JSON files.
- cleaned_weather_dataset.csv created in S3 with 19 fields per day.
- AWS cost alert system tested and functional (11min alert latency).
- 7 days of data available, with multiple records per station.
- Scaling up to fetch 6–12 months and multiple locations.

VI. Roadmap

Week	Objective
Week 8	Expand data collection to full year of historical data from 5+ weather stations
Week 9	Finalize dataset format and complete feature engineering pipeline
Week 10	Train and tune LSTM forecasting model on SageMaker
Week 11	Implement real-time alert system using Lambda and SNS
Week 12	Develop interactive dashboard (Power BI or Streamlit)
Week 13	Final integration testing and submission of all project deliverables

VII. Obstacles & Backup Plans

- **NOAA API rate limits**
  We have observed rate limiting from the NOAA API. Our backup plan involves implementing exponential retry/backoff logic to handle these limits gracefully.
- **Limited training data**
  Currently we only have temporary access to limited historical data. If this persists, we'll supplement with batch history dumps or implement a simpler regression model.
- **Cost of SageMaker GPU**
  GPU costs are currently being evaluated. As a fallback, we can switch to CPU instances or conduct local training if needed.
- **Visualization choice**
  The visualization platform remains undecided. Streamlit will serve as our default option if Power BI integration proves problematic.
- **AWS cost overruns**
  We're actively monitoring costs and will implement auto-scaling rules if expenditures approach our budget thresholds.

VIII. Expected Deliverables

By the project's completion, we will deliver a comprehensive weather forecasting system with the following components:

- **Fully Functional Data Pipeline**
  A seamless system that collects real-time weather data, processes it efficiently, and prepares it for analysis—ensuring accurate and up-to-date forecasts.
- **Cleaned and Structured Dataset**
  A high-quality historical weather dataset in CSV format, carefully organized and error-free, ready for use in predictive modeling.
- **Cost Monitoring for AWS Resources**
  A budget-friendly tracking system that alerts the team if cloud service expenses approach predefined limits.
- **Deployed Weather Prediction Model**
  A machine learning model capable of forecasting weather patterns, hosted on a reliable cloud platform.
- **Real-Time Weather Alerts**
  Automated notifications for extreme weather conditions delivered via email or messaging services.
- **User-Friendly Forecast Dashboard**
  An interactive display where users can explore weather predictions through charts and maps.
- **Complete Project Documentation**
  A well-documented code repository and final report with clear instructions for future improvements.

IX. References

1) NOAA Technical Report NWS 32 (2022)
2) AMS Journal of Atmospheric Computing (2021)