ProjectPro

Build a real-time Streaming Data Pipeline using Flink and Kinesis

CookBook

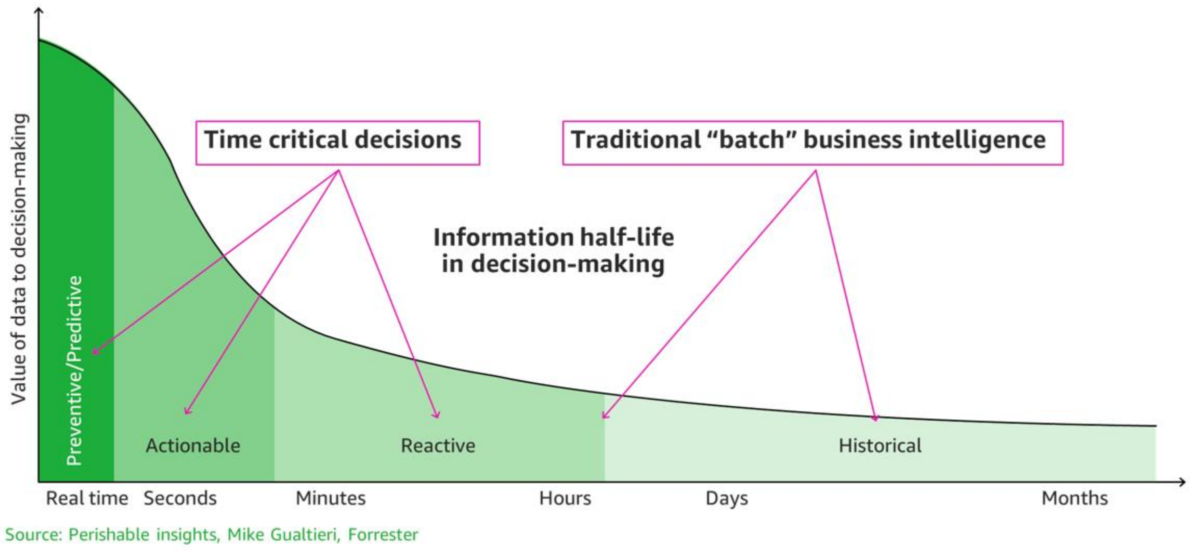
Table of Content:

Introduction

In this training, we shall look into a pipeline that serves real-time data with the highest value as the data point is generated, corresponding to which an actionable item needs to be initiated. The use cases for the proposed pipeline can be as follows:

- A ticketing system through call centers, chat-bot, etc., that carries a priority level and can be assigned to the concerned department in real-time.
- A unified system for all government agencies and departments for handling inquiries and prioritizing the ones received, irrespective of the influx of data and trends.
- For disaster/hazard management, wherein at the time of emergencies, the organization held in charge can bifurcate the distress calls based on severity and allocate resources accordingly. The goal is to minimize latency to avoid casualties.

The above examples are associated with 'Perishable Data' whose value can significantly decline over a period of time. When the operating conditions change to the point that the knowledge is no longer helpful, the information loses its value. This type of information is time-critical, which means they carry a window for preventive/predictive actions to be taken. On the other hand, Reactive data, which is historical and in 'Batch,' can be used for development in areas such as business intelligence.



- Batch vs. Stream Data

	Batch Processing	Stream Processing
Data Scope	Queries or processing over all or most data in the dataset.	Queries or processing over data within a rolling time window or on the most recent data record.
Data Size	Large batches of data.	Individual records or micro batches consisting of a few records.
Performance	Latencies in minutes to hours.	Requires latency in the order of seconds or milliseconds.
Analysis	Complex Analytics	Simple response functions, aggregates, and rolling metrics.

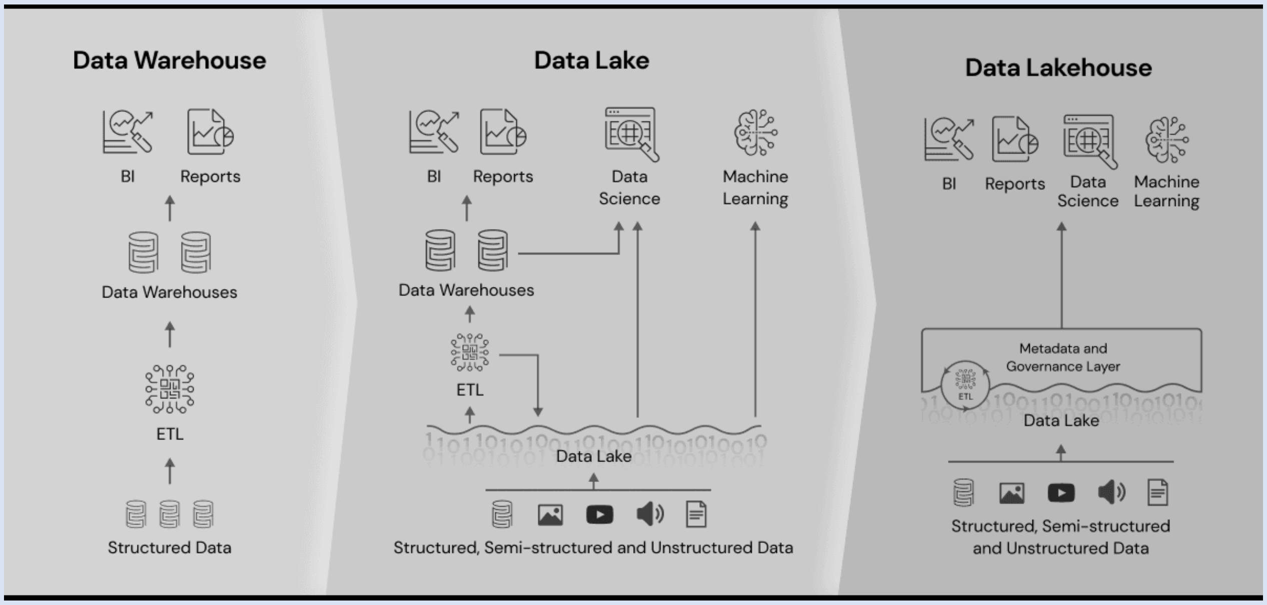
- Examples of Real-Time data processing
  - F1 race- The sensors in the Car send the Geo-location, Engine Temperature, etc., to data centers. Accordingly, the driver is instructed to pit stop.
  - Managing logistics for a Delivery Company
  - Autonomous Vehicles process image data and other sensor data within milliseconds.

- The characteristic of time-critical data can be listed as follows:
  - Staging the data based on the Availability of Data
  - Consistency in storage and lakehouse solutions
  - Partitioning the data on an evenly distributed metric
  - Effective ingestion of data in the analytical infrastructure of the organization
  - Designing various streams based on the pre-defined actions to be taken
  - Scalability
  - Effective reporting using methods such as visualization
  - Real-time decision and scheduling based on priority

The type of data suitable for this pipeline can originate from an IoT device, server logs, stand-alone applications, or APIs. They can be delimiter-separated data points, JSON-like objects, or any other form of raw data generated versus time in small sizes.

Did you know?

To enable business intelligence (BI) and machine learning (ML) on all data, a new, open data management architecture called a "data lakehouse" combines the size, flexibility, and cost-effectiveness of data lakes with the data management and ACID transactions of data warehouses. A novel, open system design that implements comparable data structures and data management capabilities to those in a data warehouse directly on the kind of inexpensive storage used for data lakes makes it possible to create data lakehouses. When combined into one system, data teams can work more quickly since they can use data without visiting numerous systems. Additionally, data lakehouses guarantee that teams working on data science, machine learning, and business analytics initiatives have access to the most complete and current data accessible.



Use-case depicted in this project

This project will simulate real-time accident data and architect a pipeline to help us analyze and take quick actions using AWS Kinesis, Apache Flink, Grafana, and Amazon SNS.
Real-time systems are usually designed on constant patterns and can't scale with peaks in the load resulting in latency and server failures. The services used in this project are highly scalable, on-demand, and cost-effective.

This Project uses the US accidents dataset, which includes a few of the following fields:

- Severity
- Start_Time
- End_Time
- Location
- Description
- City
- County

For demo purposes, the above data is loaded in a single CSV of ~500MB. The singular data points will be simulated as actual incidents and generated sequentially in mini-batches per second.
The pipeline needs to be optimized in terms of latency in processing and data availability, as emergencies such as car accidents can lead to severe consequences if immediate action is not taken.

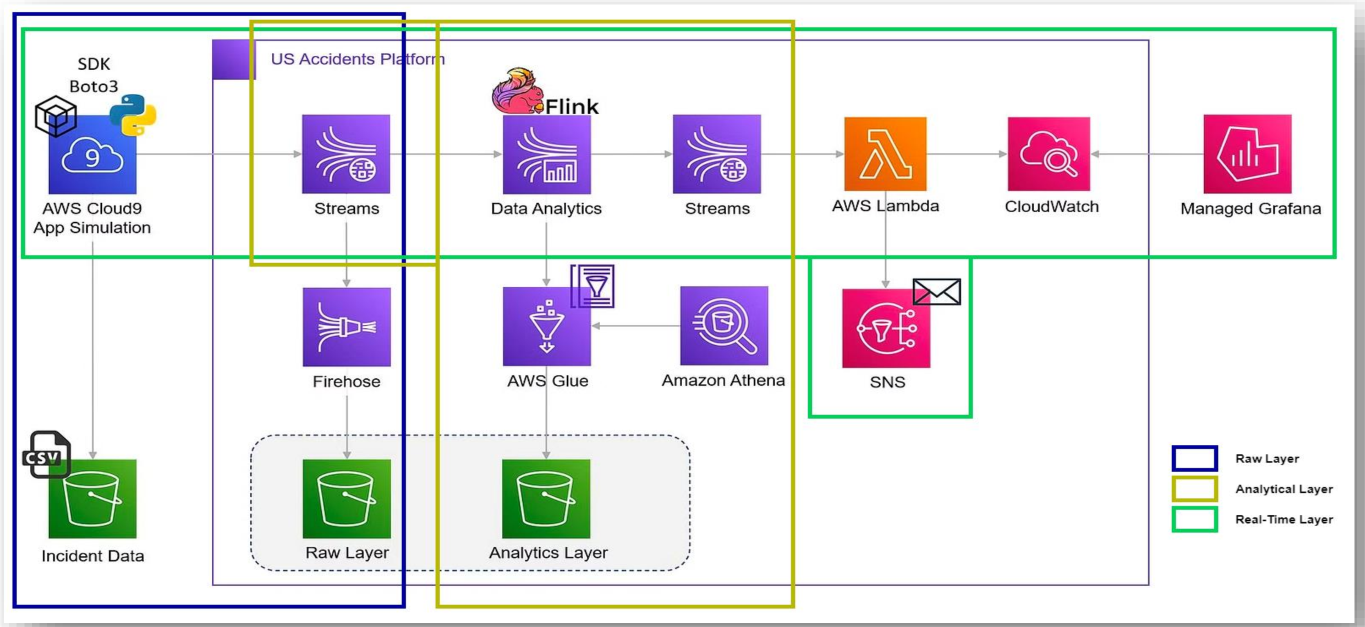
The goal of this project is to have three endpoints:

- A Raw layer- Part of the lakehouse where the raw data is streamed directly and serves as the single source of truth (SSOT).
- Analytical layer- This layer stores the filtered and transformed raw data using Apache Flink into the lakehouse for future reporting, BI applications, and developing data science applications on top of it.
- Real-time layer- The layer serves the cases over a specific severity threshold and is reported via a Grafan Dashboard. In Grafana, graphs are formed with the help of logs or metrics from any source. Alerts can be created easily in Grafana with the help of metrics and the graphs formed. Notifications are sent quickly to the concerned departments in this layer.

Did you Know?

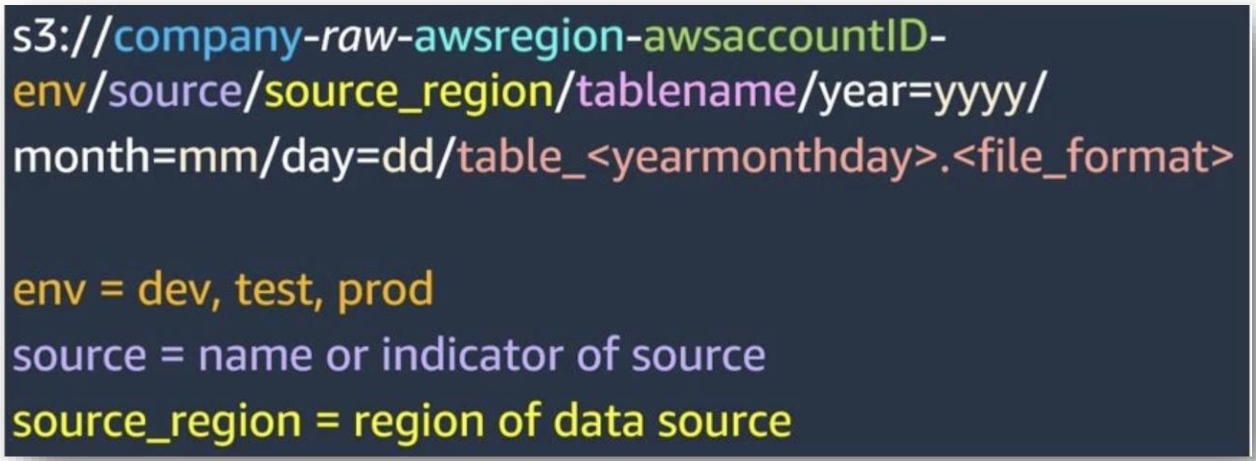
A single source of truth (SSOT) is the practice of aggregating data from many systems within an organization to a single location. An SSOT is not a system, tool, or strategy but a state of being for a company's data in that it can all be found via a single reference point.

The Architecture of the pipeline is as follows:



Breakdown of the Use-Case

- Setup required:
  - Create an AWS account
  - Setup AWS CLI
  - Download the dataset
  - Download the Code
  - This is an OS-independent project, as the whole pipeline is set up on AWS Cloud
- Ways to interact with the AWS services:
  - Console- Using the Web-based UI
  - CLI- Using the AWS CLI tool in Terminal/CMD
  - CDK- Using infrastructure-as-a-code, e.g., CloudFormation
  - SDK- Programmatically, e.g., 'boto3' in python
  - For this project, we will not be using the CDK(CloudFormation); instead, the rest of the three methods will be used for execution.
- Best Practices for AWS Account
  - Enable Multi-Factor Authentication(MFA)
  - Protect the account using IAM policies
  - Avoid using the root account
  - Choose the AWS region that is closest to you
  - Create a Budget and set up a notification
  - Rotate all keys and passwords periodically
  - Follow least privilege principle
  - Follow a naming convention for resources, as shown below



```
s3://company-raw-awsregion-awsaccountID-
env/source/source_region/tablename/year=yyyy/
month=mm/day=dd/table_<yearmonthday>.<file_format>

env = dev, test, prod
source = name or indicator of source
source_region = region of data source
```

Did you Know?

Give only the rights necessary to complete a task when setting permissions with IAM policies. You achieve this by specifying least-privilege permissions, commonly known as the activities that can be performed on limited resources under specific circumstances.

The Raw-Layer

Unique Services used in the Raw-Layer

AWS S3

Organizations need help finding, storing and managing your data. Running applications, delivering content to customers, hosting high-traffic websites, and backing up emails and other files all necessitated a significant amount of storage. Maintaining the organization's repository was costly and time-consuming for various reasons.

Among the difficulties were the following:

- Purchasing hardware and software components
- Maintenance necessitates the involvement of a team of experts.
- Due to the requirements, there needs to be more scalability.
- Requirements for data security

Amazon S3 is a cloud storage service that works differently from other popular cloud storage providers. With Amazon S3, data is stored, protected, and retrieved from "buckets" that you can access anywhere, from any device. You store "objects," such as photos, videos, documents, or other files, in the cloud, and each account can have up to 100 buckets (though you can also request a service limit increase if needed). There's no limit to the number of files you can store in each bucket.

Advantages of AWS S3:

- S3 has a durability of 99.99 percent.
- It offers a variety of "storage classes" for storing data. These classes are based on the frequency and speed with which files must be accessed.
- S3 only charges you for the resources you use, with no hidden fees or overages. You can quickly scale your storage resources to meet the ever-changing needs of your organization.
- It provides a comprehensive set of access management tools and encryption features to ensure maximum security.
- It is ideal for many applications, including data storage, backup, software delivery, data archiving, disaster recovery, website hosting, mobile applications, IoT devices, and more.

Disadvantages of AWS S3:

Every cloud storage service has cons, and Amazon S3 is no exception. Here are some of the main disadvantages of Amazon S3:

- Operations on directories are potentially slow and non-atomic.
- Not all file operations are supported. In particular, some file operations needed by Apache HBase are unavailable, so HBase cannot be run on top of Amazon S3.
- Data is visible in the object store once the entire output stream has been written.

- For any "serious" support, they require the "AWS Support Plan," which it's billed separately (around 29 USD per month)
- Downloading data, it's a bit expensive, 0,09 USD per GB after the first transferred GB/month.
- For beginners using the web interface, it might be "not so intuitive" to set up bucket permissions as it requires configuring additional AWS services like IAM.
- Even if both services don't have the same set of capabilities, prices in Backblaze B2 are far lower (around 0.005 USD per GB/month)
- It has a complex pricing schema. Even more complicated when we use the Glacier storage class that has both "retrieval time" and "cost per request.

Alternatives:

- Google Cloud Storage
- Azure Blob Storage

AWS Cloud9

AWS Cloud9 is a browser-based integrated development environment (IDE) that allows you to write, run, and debug code. It comes with a code editor, a debugger, and a terminal. Cloud9 has essential tools for popular programming languages such as JavaScript, Python, PHP, and others. To begin new projects, you do not need to install files or configure your development machine. Because your Cloud9 IDE is cloud-based, you can work on your projects from your office, home, or wherever you have an internet connection. Cloud9 also offers a unified experience for developing serverless applications, allowing you to easily define resources, debug, and switch between local and remote execution. In the development environment, you can use Cloud9 to track each other’s inputs in real-time.

Advantages of Cloud9:

- It provides a highly intuitive and easy-to-use interface, so adapting to it takes little time.
- It contains the most common programming languages.
- It allows the export of files directly to the repository or to the team where you want them.
- It is housed in the cloud so that you can work with it from anywhere.
- Allows you to share code or work on them with others in real-time.
- Cloud9 Does not require installation.

Alternatives:

- Eclipse.
- Visual Studio.
- IntelliJ IDEA.
- PyCharm.
- Codenvy.
- The Jupyter Notebook.
- NetBeans.
- PhpStorm.

Amazon Kinesis

Real-time streaming data can be easily collected, processed, and analyzed with Amazon Kinesis to gain immediate insights and respond rapidly to new information. With the freedom to select the tools that best meet your application's needs, Amazon Kinesis provides essential features for processing streaming data at any scale economically. You may ingest real-time data for machine learning, analytics, and other applications with Amazon Kinesis, including video, audio, application logs, website clickstreams, and IoT telemetry data. Instead of waiting until all your data has been collected before processing, Amazon Kinesis lets you process and analyze data as it comes in and responds immediately.

Services under Kinesis:

- **Kinesis Data Streams**
  - Collect and Store Data Streams for analysis
  - Streams stores data in 'Shards' with each shard serving 1MB/s OR 1000 records/s. The field selected for shards(partitions) should be balanced.
  - The data retention period is 24hrs by default, which can be increased for up to 7 days and further increased upon request to AWS.
  - When the data stream throughput is unpredictable and auto-scaling is required, one can use the 'On-Demand' configuration. This configuration provides a writing capacity of a maximum of 200 MB/s OR 200,000 records/s and a read capacity of 400 MB/s
  - If the throughput increases more than 200 MB/s or better control is needed over the number of shards created, the 'Provision' configuration can be selected.
  - For the pricing of Data Streams, click here.

- **Kinesis Data Firehose**
  - Amazon Kinesis Data Firehose is a fully managed service to deliver real-time streaming data to destinations such as Amazon S3 and Redshift.
  - You don't need to write applications or manage resources with Kinesis Data Firehose. You configure your data producers to send data to Kinesis Data Firehose, and it automatically delivers the data to the destination that you specified. You can also configure Kinesis Data Firehose to transform your data before returning it.
  - The data of interest that your data producer sends to a Kinesis Data Firehose delivery stream. A record can be as large as 1,000 KB.
  - No facility for data storage.
  - Near real-time processing capabilities, depending on the buffer size or minimum buffer time of 60 seconds.

- **Kinesis Data Analytics**
  - The simplest method to analyze streaming data, generate practical insights, and instantly address customer and company needs is through Amazon Kinesis Data Analytics. The complexity of developing, running, and connecting streaming applications with other AWS services is decreased with Amazon Kinesis Data Analytics.
  - You can quickly build SQL queries and sophisticated Apache Flink applications in a supported language such as Java or Scala using built-in templates and operators for standard processing functions to organize, transform, aggregate, and analyze data at any scale.

- Everything needed to run your real-time applications constantly is handled by Amazon Kinesis Data Analytics, which scales automatically to match the volume and throughput of your incoming data. You only pay for the resources your streaming applications use using Amazon Kinesis Data Analytics. There are no setup fees or minimum charges.

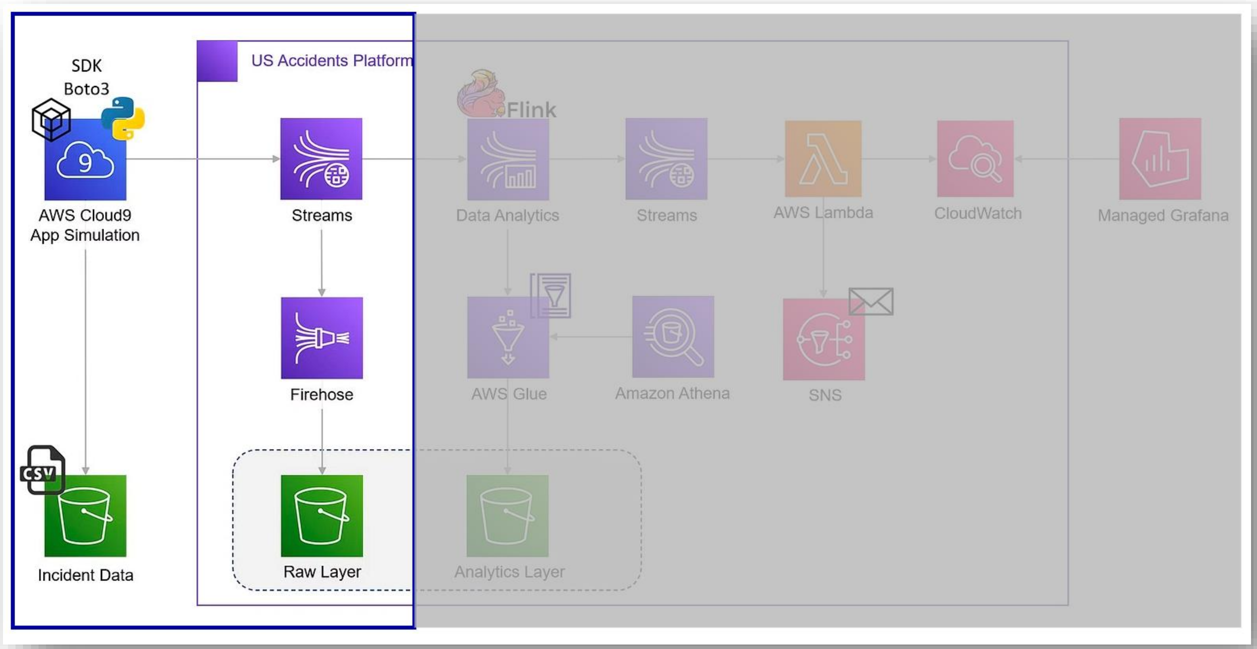
Advantages of Amazon Kinesis:

- Most analytics tools either delay processing, sample data across the stream, or process streaming data in chunks. Amazon Kinesis Data Analytics performs processing on the data stream in real time.
- You can gain real-time insights from machine learning on streaming data. Your time to insights will be almost immediate because of Kinesis Data Analytics' processing power.
- Because Amazon Kinesis Data Analytics is serverless, you don't have to bother setting up and managing servers; the underlying infrastructure is handled and automated in the background, allowing you to concentrate on your data analytics rather than server maintenance.
- In most cases, Kinesis Data Analytics elastically adjusts your application to consider your source stream's data throughput and query complexity. By setting up the parallel execution of tasks and resource allocation, scaling can be enabled.
  Assume you bundle your incoming data streams using AWS Kinesis Data Analytics and Amazon Kinesis Data Streams. Then, it conveniently enables you to perform standard processing across the board, and with this setup, there is no data transmission price. The records for Data Streams are chunked into 25KB units, and each shard can handle 1MB/sec or 1000 PUT records per second. Shards are made accessible by the hour, and up to 2 MB/s can be produced. Stream data are typically accessible for up to 24 hours.
- The Amazon Kinesis Data Analytics service uses a pay-as-you-go business model, like many other AWS services. As a result, no wasted resources or tasks are required to optimize the underlying resources; you only pay for what you utilize (beyond the streaming data you throughput).
- No need to master brand-new programming languages, frameworks, or machine learning techniques.
  With Amazon Kinesis Data Analytics, the learning curve for new products is considerably shortened because you don't need to learn any new languages or frameworks to get started. You may quickly put together your preferred data analytics operations with the help of the service, which contains machine learning algorithms and preconfigured SQL procedures.

Alternatives:

- Google Cloud Dataflow.
- Google Cloud Pub/Sub.
- Spark Streaming.
- Apache Flink.
- Azure Event Hubs.
- Azure Stream Analytics.
- Apache Kafka

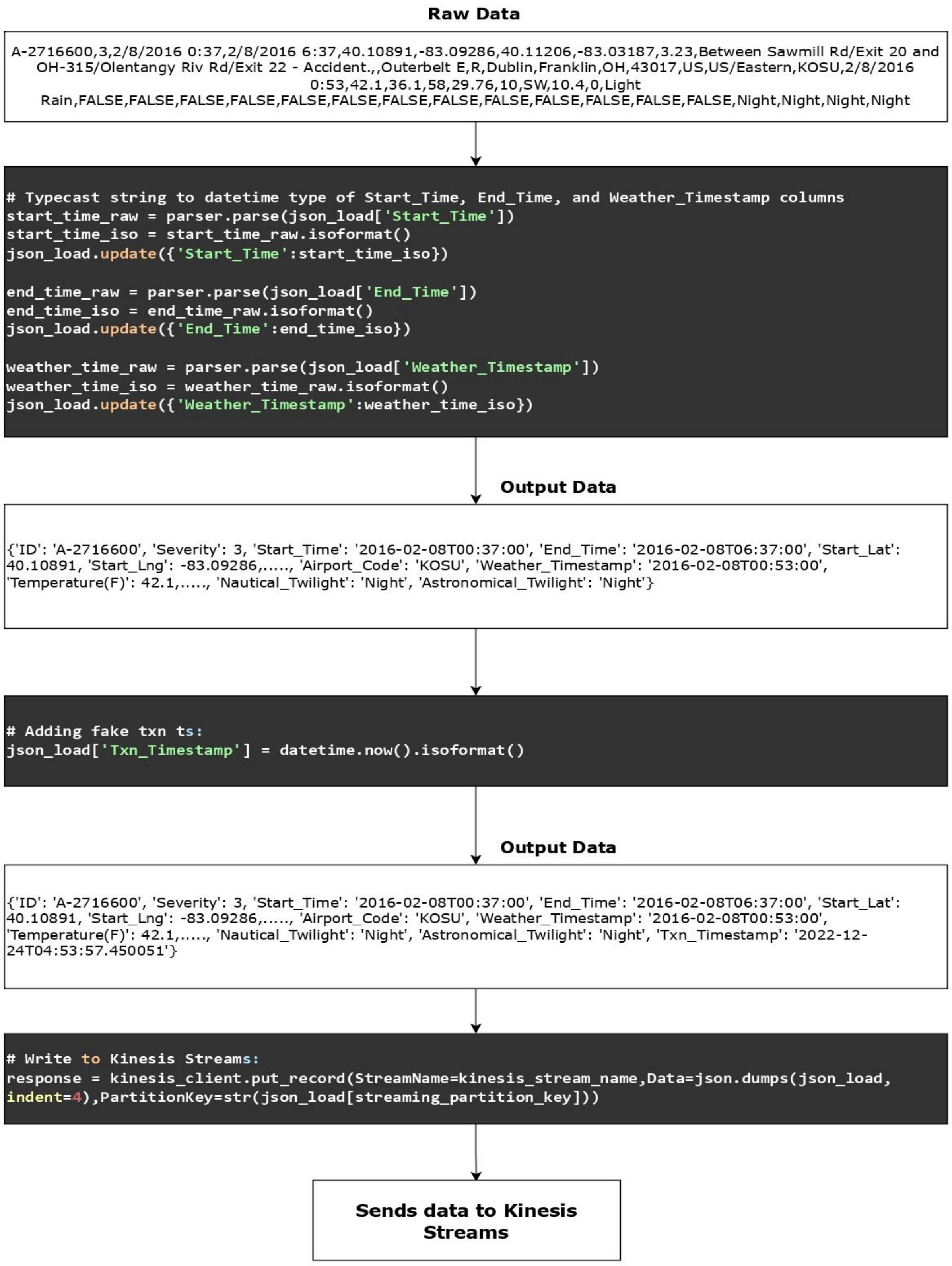
Raw-Layer



Steps in Raw-Layer

1. Create an S3 bucket
2. Create Data Stream named 'us-accidents-data-stream-1'
3. Upload RAW data file in S3
4. Setup Cloud9 environment and run the simulation Code
    a. Read the file from S3
    b. Convert each row to JSON
    c. Typecast strings to Datetime object
    d. Add a Transaction Timestamp (Txn_Timestamp)
    e. Push each data point to 'us-accidents-data-stream-1'
5. After implementing the Analytical and the Real-time Layer, push the Raw data to S3 as part of the SSOT

Code Flow



Raw Data

```
A-2716600,3,2/8/2016 0:37,2/8/2016 6:37,40.10891,-83.09286,40.11206,-83.03187,3.23,Between Sawmill Rd/Exit 20 and
OH-315/Olentangy Riv Rd/Exit 22 - Accident.,,Outerbelt E,R,Dublin,Franklin,OH,43017,US,US/Eastern,KOSU,2/8/2016
0:53,42.1,36.1,58,29.76,10,SW,10.4,0,Light
Rain,FALSE,FALSE,FALSE,FALSE,FALSE,FALSE,FALSE,FALSE,FALSE,FALSE,FALSE,FALSE,FALSE,Night,Night,Night,Night
```

```
# Typecast string to datetime type of Start_Time, End_Time, and Weather_Timestamp columns
start_time_raw = parser.parse(json_load['Start_Time'])
start_time_iso = start_time_raw.isoformat()
json_load.update({'Start_Time':start_time_iso})

end_time_raw = parser.parse(json_load['End_Time'])
end_time_iso = end_time_raw.isoformat()
json_load.update({'End_Time':end_time_iso})

weather_time_raw = parser.parse(json_load['Weather_Timestamp'])
weather_time_iso = weather_time_raw.isoformat()
json_load.update({'Weather_Timestamp':weather_time_iso})
```

Output Data

```
{'ID': 'A-2716600', 'Severity': 3, 'Start_Time': '2016-02-08T00:37:00', 'End_Time': '2016-02-08T06:37:00', 'Start_Lat':
40.10891, 'Start_Lng': -83.09286,....., 'Airport_Code': 'KOSU', 'Weather_Timestamp': '2016-02-08T00:53:00',
'Temperature(F)': 42.1,....., 'Nautical_Twilight': 'Night', 'Astronomical_Twilight': 'Night'}
```

```
# Adding fake txn ts:
json_load['Txn_Timestamp'] = datetime.now().isoformat()
```

Output Data

```
{'ID': 'A-2716600', 'Severity': 3, 'Start_Time': '2016-02-08T00:37:00', 'End_Time': '2016-02-08T06:37:00', 'Start_Lat':
40.10891, 'Start_Lng': -83.09286,....., 'Airport_Code': 'KOSU', 'Weather_Timestamp': '2016-02-08T00:53:00',
'Temperature(F)': 42.1,....., 'Nautical_Twilight': 'Night', 'Astronomical_Twilight': 'Night', 'Txn_Timestamp': '2022-12-
24T04:53:57.450051'}
```

```
# Write to Kinesis Streams:
response = kinesis_client.put_record(StreamName=kinesis_stream_name,Data=json.dumps(json_load,
indent=4),PartitionKey=str(json_load[streaming_partition_key]))
```

Sends data to Kinesis Streams

Analytical layer

Unique Services used in the Analytical Layer

Apache Flink:

Apache Flink is an open-source, unified stream-processing and batch-processing framework developed by the Apache Software Foundation. The core of Apache Flink is a distributed streaming dataflow engine written in Java and Scala and executes arbitrary dataflow programs in a data-parallel and pipelined manner. Flink is used for stateful computation over unbounded and bounded data streams and performs analysis at an in-memory speed.

Alternatives:

- Azure Stream Analytics
- Google Dataflow
- TIBCO Spotfire
- StreamSets
- Azure Event Hubs
- Google Cloud Pub/Sub

Did you Know?
The top 5 Apache Projects by Commits are

1. *Camel*
2. *Flink*
3. *Airflow*
4. *Lucene/Solr*
5. *Spark*

Did you Know?
Stateful Functions is an API that simplifies building distributed stateful applications. It's based on functions with a persistent state that can interact dynamically with strong consistency guarantees.

AWS Glue:

Customers can easily prepare and load their data for analytics with AWS Glue, a fully managed extract, transform, and load (ETL) service. The AWS Management Console allows you to design and execute an ETL process in just a few clicks. You merely direct AWS Glue to the data that is kept on AWS. Your data is crawled by Glue, adding related metadata (such as the table definition and schema) to the AWS Glue Data Catalog. Your data is immediately searchable, query-able, and ETL-ready after it has been cataloged. Here are the features that make AWS Glue useful:

AWS Glue Data Catalog

Effective data queries and transformations can be created using the AWS Glue Data Catalog. The data catalog is a repository for the metadata associated with the data you wish to work with. It also automatically registers partitions, maintains a history of data schema modifications, and retains other control data related to the ETL environment. It also includes definitions of processes and data tables. AWS Glue can function as it does because of the data catalog, which is a necessary component.

Automatic ETL Code Generation

Automatic ETL code generation is one of the most noticeable characteristics. AWS Glue will build the Python or Scala code for the ETL pipeline after the user specifies the data source and destination. The code can parallelize demanding workloads and is compatible with Apache Spark.

Automated Data Schema Recognition

AWS Glue can automatically recognize the schema of your data. Crawlers that parse the data and its sources or targets are used to accomplish this. Due to the numerous intricate details of the ETL processes and characteristics of data sources and destinations, it is practical since users are spared from having to define the schema for their data individually.

Advantages of AWS Glue

- Infrastructure setup or maintenance is not required for the execution of ETL tasks. All of the more minute aspects are handled by Amazon. Many processes are automated via AWS Glue. The data schema can be easily ascertained, code may be generated, and customization can begin. Logging, monitoring, alerting, and restarting in failure scenarios are all made simpler by AWS Glue.
- Complements other AWS tools effectively. As a result, AWS Glue is relatively simple to integrate with data sources and targets like Amazon Kinesis, Amazon Redshift, Amazon S3, and Amazon MSK. Additionally compatible with Amazon EC2 instances are several widely used data storages.
- Since customers only pay for the resources they use, Glue might be economical. You don't have to pay for peak-time resources outside of this time if your ETL jobs occasionally need more processing power but usually use fewer resources.

Disadvantages of AWS Glue:

- Jobs are run in Apache Spark through AWS Glue. The engineers that modify the resultant ETL job must be proficient with Spark. The code will be written in Scala or Python. Thus developers should also be familiar with those two languages. Not all data professionals have the skills to customize ETL jobs for their unique requirements.
- Spark struggles to handle joins with high cardinality. But they are essential for other use cases, including fraud detection, advertising, gaming, etc. To make such joins effective, additional activities will be required. Implement a second database, for instance, to keep track of transient states. ETL pipelines become more difficult as a result.
- Combining stream and batch is challenging. Combining stream and batch processing with Glue is feasible, but it is a complex procedure. Stream and batch operations must be kept apart to use AWS Glue. This implies that the same code must be created and improved twice. A developer should also make sure that stream and batch processes do not conflict with one another.
- Although Glue is designed to operate with other AWS services, it does not integrate with third-party products, which may restrict your options when creating an open lake architecture.

Alternatives:

- Talend Data Integration
- Azure Data Factory
- Apache NiFi

AWS Athena:

Data in Amazon S3 can be easily analyzed using regular SQL, thanks to Amazon Athena, an interactive query service. Because Athena is serverless, you only pay for the queries you perform, and there is no infrastructure to maintain. Utilizing Athena is simple: specify the schema, point to your data in Amazon S3, and begin basic SQL queries. Most findings arrive in a matter of seconds. To prepare your data for analysis, Athena eliminates the requirement for challenging extract, transform, and load (ETL) tasks. This makes it simple for anyone with SQL knowledge to quickly analyze massive datasets. With Athena's out-of-the-box integration with AWS Glue Data Catalog, you can build a consistent metadata repository for use by different services, crawl data sources to find schemas, add new and updated table and partition definitions to your Catalog, and keep track of schema versioning.

Advantages of AWS Athena:

- Serverless: AWS Athena spares you all the hassle associated with infrastructure maintenance because it is delivered as a fully-managed serverless service. Setting up clusters, managing capacity, or loading data is not an issue.
- Cost-effective: Far less expensive than its closest rival is AWS Athena. The service doesn't charge you for compute instances, which is the cause. You only pay for the queries you conduct.
- Widely accessible: AWS Athena is available to everyone, not just developers and engineers, as it conducts its queries using standard SQL. Since routine SQL queries are relatively basic and easy to understand, business analysts and other data professionals can embrace them.
- Flexibility: Thanks to Amazon Athena's open and adaptable architecture, you aren't tied to a particular manufacturer, technology, or tool. For instance, you can freely switch between query engines without modifying the schema and work with various open-source file types.

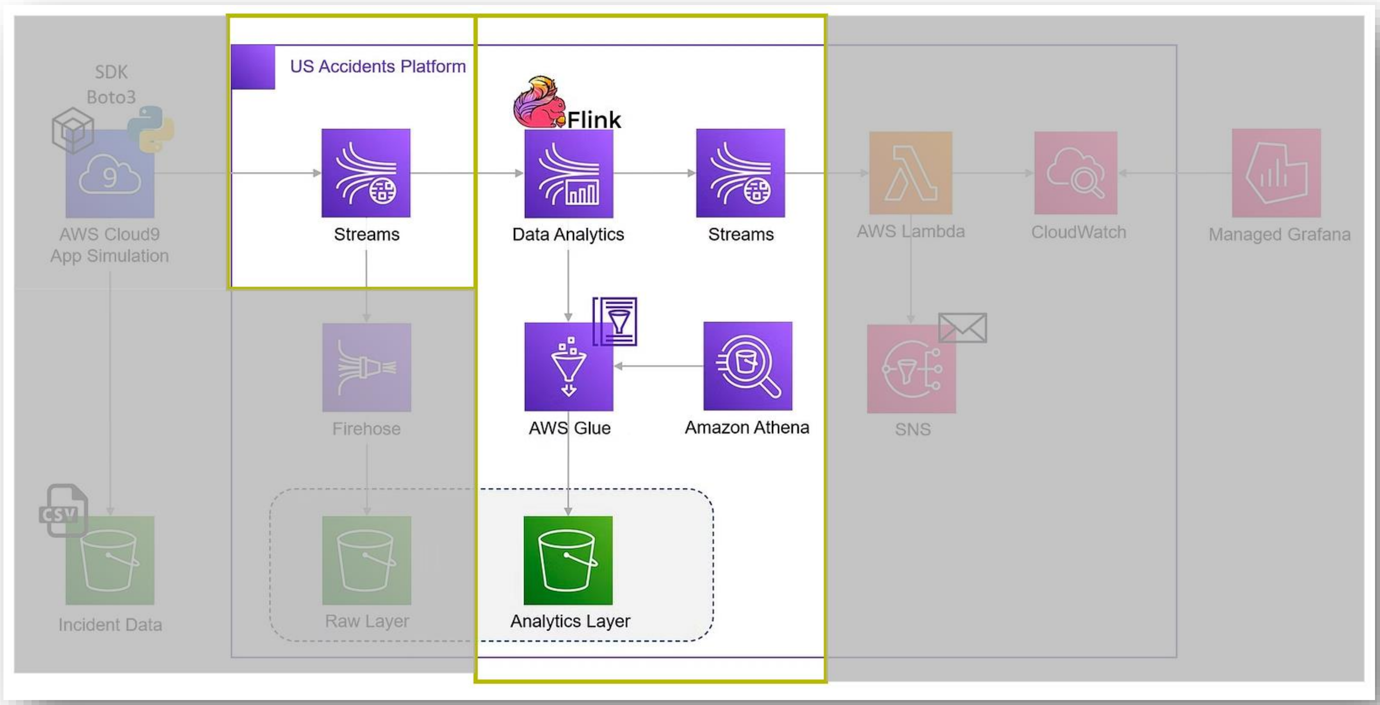
Disadvantages of AWS Athena:

- There aren't many optimization options available with AWS Athena. The most you can do here is query optimization, not underlying data optimization. You must still be cautious while using AWS Glue to transform Amazon S3 data to protect other services that access the same data.
- All AWS Athena customers worldwide are required to use the same resources to conduct their queries per the Service Level Agreement (SLA) of Amazon. This multi-tenancy strategy could occasionally cause resource stress, resulting in variable query performance.
- All you'll find here is a query engine because AWS Athena is only a query service. It lacks a Data Manipulation Language (DML) interface that allows for data insertion, deletion, and updation.
- Consider partitioning the datasets kept in Amazon S3 if you want to conduct your SQL queries effectively. The number of divisions you can construct will significantly impact how quickly and effectively your queries run. For instance, your querying time will increase by a second for every 500 partitions inspected.
- While indexing has long been a standard feature in conventional databases, AWS Athena does not grant you this privilege. Therefore, you should anticipate difficulties when doing procedures like aggregating big tables.

Alternatives:

- AWS RDS
- Oracle Database
- Databricks Lakehouse Platform

Analytical Layer



Steps in Analytical Layer

1. Test the Flink application in SQL or Python code provided
2. Build and deploy the Flink application in Kinesis Data Analytics(KDA)
3. The Flink Application includes:
    a. Create a table in the Glue database and ingest the 'us-accidents-data-stream-1' data via Glue Catalog
    b. Watermark on Txn_Timestamp with 5-second interval
    c. Partition the data based on the 'Severity' field
    d. Create a second table for the Real-time layer by filtering the following fields and pushing them to 'us-accidents-data-stream-2':
        i. ID
        ii. Severity
        iii. City
        iv. County
        v. Txn_Timestamp

Code Flow

/* 1. Create a table to store data from the 'us-accidents-data-stream-1' kinesis stream into the Glue database.*/

```
DROP TABLE IF EXISTS us_accidents_stream;
CREATE TABLE us_accidents_stream (
    `ID` VARCHAR(50),
    `Severity` bigint,
    `Start_Time` TIMESTAMP(3),
    `End_Time` TIMESTAMP(3),
     ........
    `City` VARCHAR(50),
    `County` VARCHAR(50),
    ..........
    `Txn_Timestamp` TIMESTAMP(3),
    WATERMARK FOR Txn_Timestamp as Txn_Timestamp - INTERVAL '5' SECOND
    )
    PARTITIONED BY (Severity)
    WITH (
        'connector' = 'kinesis',
        'stream' = 'us-accidents-data-stream-1',
        'aws.region' = 'eu-west-1',
        'scan.stream.initpos' = 'LATEST',
        'format' = 'json',
        'json.timestamp-format.standard' = 'ISO-8601'
    );
```

/* 2. Create a table to store only the selective columns and send it to 'us-accidents-data-stream-2'*/

```
DROP TABLE IF EXISTS us_accidents_stream_1_results;
CREATE TABLE us_accidents_stream_1_results (
    `ID` VARCHAR(50),
    `Severity` bigint,
    `City` VARCHAR(50),
    `County` VARCHAR(50),
    `Txn_Timestamp` TIMESTAMP(3)
    )
    PARTITIONED BY (Severity)
    WITH (
      'connector' = 'kinesis',
      'stream' = 'us-accidents-data-stream-2',
      'aws.region' = 'eu-west-1',
      'format' = 'json',
      'json.timestamp-format.standard' = 'ISO-8601'
);
```

/* 3. Send only the selective data from 'us_accidents_stream' table to 'us_accidents_stream_1_results' table*/

```
insert into us_accidents_stream_1_result6
select  ID, Severity, City,  County, Txn_Timestamp
from us_accidents_stream where Severity > 3;
```

Did you Know?

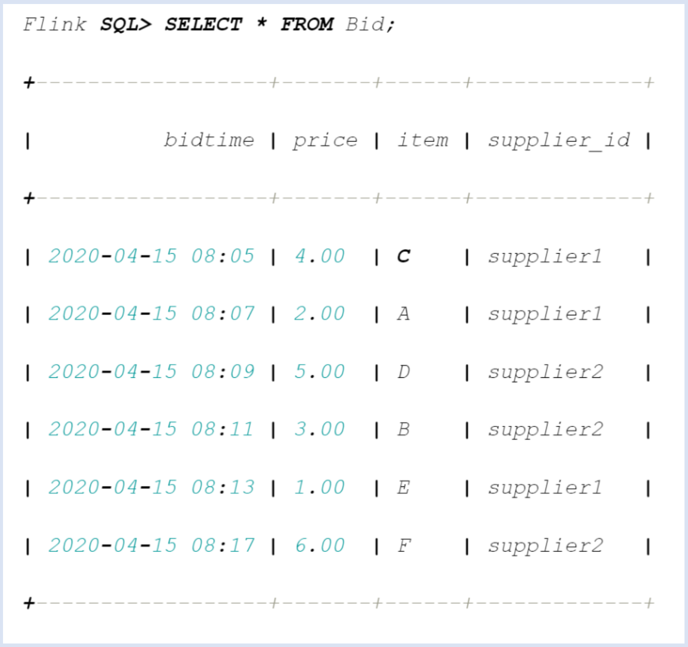
Enhanced fan-out is an Amazon Kinesis Data Streams feature that enables consumers to receive records from a data stream with dedicated throughput of up to 2 MB of data per second per shard. A consumer using enhanced fan-out doesn't have to contend with other consumers receiving data from the stream.

For a streaming application of unbounded data sets, the completeness of all incoming data is crucial. To guarantee that every data is processed, you can use watermarks in Flink applications to track the events within a given timeframe(window).

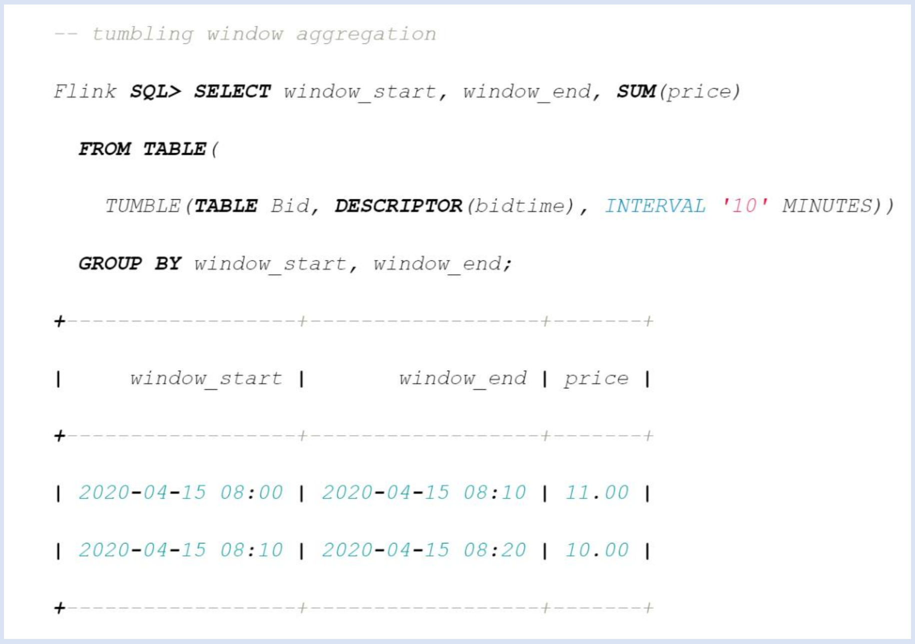
Did you Know?

When a windowed query processes each window in a non-overlapping manner, the window is referred to as a tumbling window. In this case, each record on an in-application stream belongs to a specific window. It is processed only once (when the query processes the window to which the record belongs).

Example-



```
Flink SQL> SELECT * FROM Bid;

+------------------+-------+------+-------------+

|          bidtime | price | item | supplier_id |

+------------------+-------+------+-------------+

| 2020-04-15 08:05 | 4.00  | C    | supplier1   |

| 2020-04-15 08:07 | 2.00  | A    | supplier1   |

| 2020-04-15 08:09 | 5.00  | D    | supplier2   |

| 2020-04-15 08:11 | 3.00  | B    | supplier2   |

| 2020-04-15 08:13 | 1.00  | E    | supplier1   |

| 2020-04-15 08:17 | 6.00  | F    | supplier2   |

+------------------+-------+------+-------------+
```



```
-- tumbling window aggregation

Flink SQL> SELECT window_start, window_end, SUM(price)

  FROM TABLE(

    TUMBLE(TABLE Bid, DESCRIPTOR(bidtime), INTERVAL '10' MINUTES))

  GROUP BY window_start, window_end;

+------------------+------------------+-------+

|     window_start |       window_end | price |

+------------------+------------------+-------+

| 2020-04-15 08:00 | 2020-04-15 08:10 | 11.00 |

| 2020-04-15 08:10 | 2020-04-15 08:20 | 10.00 |

+------------------+------------------+-------+
```

Real-Time layer

Unique Services used in the Real-Time Layer

AWS Lambda:

AWS Lambda lets you run code without provisioning or managing servers. You pay only for the computing time you consume—there is no charge when your code is not running. With Lambda, you can run code for virtually any application or backend service—all with zero administration. Just upload your code, and Lambda takes care of everything required to run and scale your code with high availability. You can set up your code to trigger automatically from other AWS services or call it directly from any web or mobile app.

Advantages of AWS Lambda:

Many of the advantages of AWS Lambda relate to the advantages of adopting a serverless approach.

- Saves Time and Effort: One of the significant benefits of going serverless using AWS Lambda is the time and effort saved from creating and maintaining your application's infrastructure.
    - AWS saves a lot of effort by provisioning and managing the infrastructure your Lambda functions run on.
    - It automatically scales the instances to handle excessive load times and implements a proper logging and error-handling system. Those involved in constructing or maintaining infrastructure will reasonably understand the gravity of this advantage.
    - Not only does it save you time while building the application, but it will also save you ample time required to maintain an already established system as your application evolves and scales up.
    - Lambda ensures quicker production and greater agility in launching your product in the market, thus giving you an edge over your competition.
- Helps in building a scalable solution: AWS Lambda has become one of the most popular solutions in such a short span of time because of its ability to accommodate applications at scale and early stages.
    - For large loads of applications, AWS runs your Lambda function simultaneously with other Lambda functions. This means that you do not have to worry about clogged-up queues.
    - In addition to this, multiple instances of the same Lambda function can be provisioned and run at the same time.
- You Pay For What You Need: Another great advantage of using AWS Lambda is that you only pay for what you need. Lambda can accommodate applications with differing loads and charges you for the number of requests your Lambda functions receive and the time it takes to execute those requests per 100ms.

Disadvantages of AWS Lambda:

- Cold Start: The serverless architecture executes functions on temporarily created containers. Let's consider a simple function, such as a client registering their details. When the client submits their information to the Lambda function, AWS will create a temporary container, deploy all the dependencies involved, and run the required code. Once the request is completed, the container is destroyed. Now, if there is another request after some time, the same process will be followed. The drawback lies in the time taken by Lambda to create this temporary container. This is usually between 100 milliseconds to 2 minutes and is known as a cold start. However, there is a workaround that we recently implemented in our application. You must ping your Lambda every 5 minutes to ensure the container is not destroyed. This way, only the first request will take some time to process, and all the subsequent ones will be processed much faster without delay.
- Computational Restrictions: The functions which require a lot of processing cannot be handled by AWS Lambda. However, there is a workaround here as well. You can create a queuing process to break up your work into a series of smaller functions and execute them simultaneously using AWS Lambda.
- Vendor control and vendor lock-in: Another issue with AWS Lambda is that the third-party applications used to get are decided by AWS. This means you would have to give up much control over your application. Another pain point is vendor lock-in. Once the operations are built around a serverless application, it can be costly, time-consuming, and difficult to port your operations elsewhere.
- Issues with working in a Virtual Private Cloud: Many clients use a Virtual Private Cloud (VPC) for an extra layer of security. Using Lambda with any such function can entail some additional delay over and above the cold start mentioned above. This, too, can be dealt with using a warm start.

Alternatives:

- Google App Engine.
- Salesforce Heroku.
- Azure App Service.
- Cloud Foundry.
- Salesforce Platform.
- PythonAnywhere.
- Azure Functions.

Amazon CloudWatch

Amazon CloudWatch is a monitoring and management service built for developers, system operators, site reliability engineers (SRE), and IT managers. CloudWatch provides data and actionable insights to monitor your applications, understand and respond to system-wide performance changes, optimize resource utilization, and get a unified view of operational health. CloudWatch collects monitoring and operational data from logs, metrics, and events, providing a unified view of AWS resources, applications, and services that run on AWS and on-premises servers. You can use CloudWatch to set high-resolution alarms, visualize logs and metrics side by side, take automated actions, troubleshoot issues, discover insights to optimize your applications, and ensure they run smoothly.

Advantages of CloudWatch

- Since CloudWatch is already a part of the Amazon ecosystem, most users use it. Because of this, setting up and using CloudWatch is simpler. As of March 2022, Amazon offers a one-click CloudWatch setup option, making monitoring your resources and workloads simpler. People prefer using an AWS-native solution for proactive resource allocation, such as autoscaling and infrastructure monitoring.
- Alarms and rules are simpler to set up when using the new one-click option. With only one click, this function opens CloudWatch Application Insights. Application Insights sets up alarms to keep track of the health of your account's underlying resources and automatically finds them. The creation of dashboards that show alerts and issues is also made easier by Application Insights.
- Users can keep an eye on statistics and logs from on-premises servers, AWS resources, applications, and services using CloudWatch. You get access to system-wide visibility thanks to this capability. Although this high-level observability is fantastic, greater difficulties appear when users want to resolve particular problems or go deeper into the reason for ongoing performance issues.

Disadvantages of CloudWatch

- Users can keep an eye on statistics and logs from on-premises servers, AWS resources, applications, and services using CloudWatch. You get access to system-wide visibility thanks to this capability. Although this high-level observability is fantastic, greater difficulties appear when users want to resolve particular problems or go deeper into the reason for ongoing performance issues.
- Pricing may be among CloudWatch's more perplexing features. For queries and events, there are different fees. The metrics and dashboards you select to utilize will significantly impact the price. The number of custom events and alarms you create, The number of logs you save, and the windows you decide to keep for pricing are excessive, and cost estimation is also overly complicated. Consider estimating each service while keeping in mind that CloudWatch will be utilized for event investigation most of the time (e.g., errors, traffic spikes, security issues, etc.). It is impossible to forecast these occurrences correctly. In other words, it's never sure how much you'll pay each month, and prices can differ significantly depending on your objectives.
- Querying and retaining logs at scale becomes problematic from a management perspective. Once you reach the terabyte scale with your records (and wish to retain them beyond a short period,

such as a few days or a week), CloudWatch becomes too costly and difficult to use when diagnosing issues or identifying advanced persistent security threats. With that said, CloudWatch remains an excellent tool for monitoring your metrics.

Amazon Simple Notification Service

Amazon Simple Notification Service (Amazon SNS) is a highly available, durable, secure, fully managed pub/sub messaging service that enables you to decouple microservices, distributed systems, and serverless applications. Amazon SNS provides topics for high-throughput, push-based, many-to-many messaging. Using Amazon SNS topics, your publisher systems can fan out messages to many subscriber endpoints for parallel processing, including Amazon SQS queues, AWS Lambda functions, and HTTP/S webhooks. Additionally, SNS can fan out notifications to end users using mobile push, SMS, and email.

Advantages of Amazon SNS

These are the main benefits of using SNS in Serverless applications.

- Scalability: SNS topics scale up to any number of publishers, subscribers, and messages without needing infrastructure work. This is helpful for growing applications where the teams would rather not have to provide additional infrastructure as their usage of pub/sub model increases over time.
- Ease of setup: SNS is a fully managed service; setting it up requires zero infrastructure work. Initial ramp-up is also easy as SNS provides an HTTP API that conforms to API standards. With many subscriber types supported out of the box, topic subscribers are unlikely to require a different setup. SNS's ease of setup means you will have a faster path to a proof-of-concept application or a fully implemented solution.
- Multiple notification formats are supported: SNS supports AWS Lambda and AWS SQS notifications, mobile push notifications, HTTP(S) endpoints, email addresses, and SMS messages. These formats cover many common use cases, so you won't have to write custom code to implement a subscriber for your SNS topic.
- Integration with AWS Lambda: The native integration with AWS Lambda allows you to run a Lambda function whenever a message is published to an SNS topic. This allows for many useful workflows that process data from SNS messages, such as reacting to notifications from other systems or preparing the data for storage in S3 or Redshift.

Disadvantages of Amazon SNS

- High cost at scale. When using SNS, you pay for exactly what you use. This can be both positive and negative. The upside is that there are no monthly charges—if you use SNS for just one day of the month, you pay only for that day. On the downside, however, with a high number of topics, subscribers, and messages, the costs of SNS can eventually rise relatively high, and operating your infrastructure for an SNS-like load may be more cost-effective in such cases.
- Not much control over performance. SNS is a fully managed service; as a result, you don't get to look under the hood. This can be a problem if SNS is a part of your customer-facing response path or if you have certain SLAs to meet. While for many use cases, SNS performance is plenty good enough, at a certain stage, you will likely be able to get better and more predictable performance from a pub/sub system that you manage yourself and tweak to your needs compared to what SNS can offer at scale.

Grafana

Grafana is an open-source interactive data visualization platform developed by Grafana Labs. It allows users to see their data via charts and graphs that are unified into one dashboard for more straightforward interpretation and understanding. You can also query and set alerts on your information and metrics from wherever that information is stored, whether in traditional server environments, Kubernetes clusters, various cloud services, etc. You're then more easily able to analyze the data, identify trends and inconsistencies, and ultimately make your processes more efficient. Grafana was built on open principles and the belief that data should be accessible throughout an organization, not just to a small handful of people. This fosters a culture where data can be easily found and used by anyone who needs it, empowering teams to be more open, innovative, and collaborative.
Amazon Managed Grafana is a fully managed and secure data visualization service that can instantly query, correlate, and visualize operational metrics, logs, and traces from multiple sources. Amazon Managed Grafana makes it easy to deploy, operate, and scale Grafana, a widely deployed data visualization tool popular for its extensible data support.

Advantages of Grafana

- Open Source
- Different types of authentication and security levels
- Alert and notification system
- Possibility of new developments and API interaction
- Cloud Agnostic

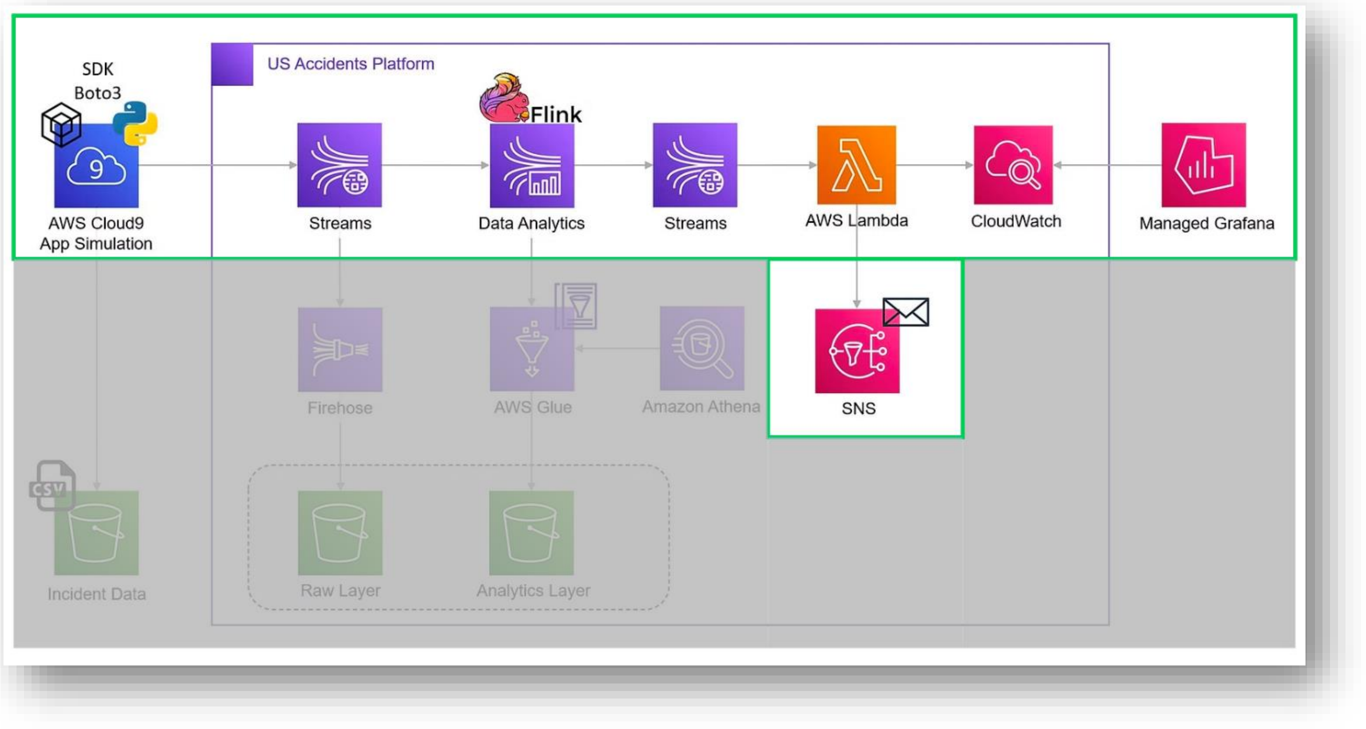
Disadvantages of Grafana

- Limits charts to time series
- Not self-managed
- Limited dashboard organization
- Limited dashboard designs
- Creation of new metrics and modeling concerning other tools

Alternatives

- Datadog
- Holistics
- Kibana

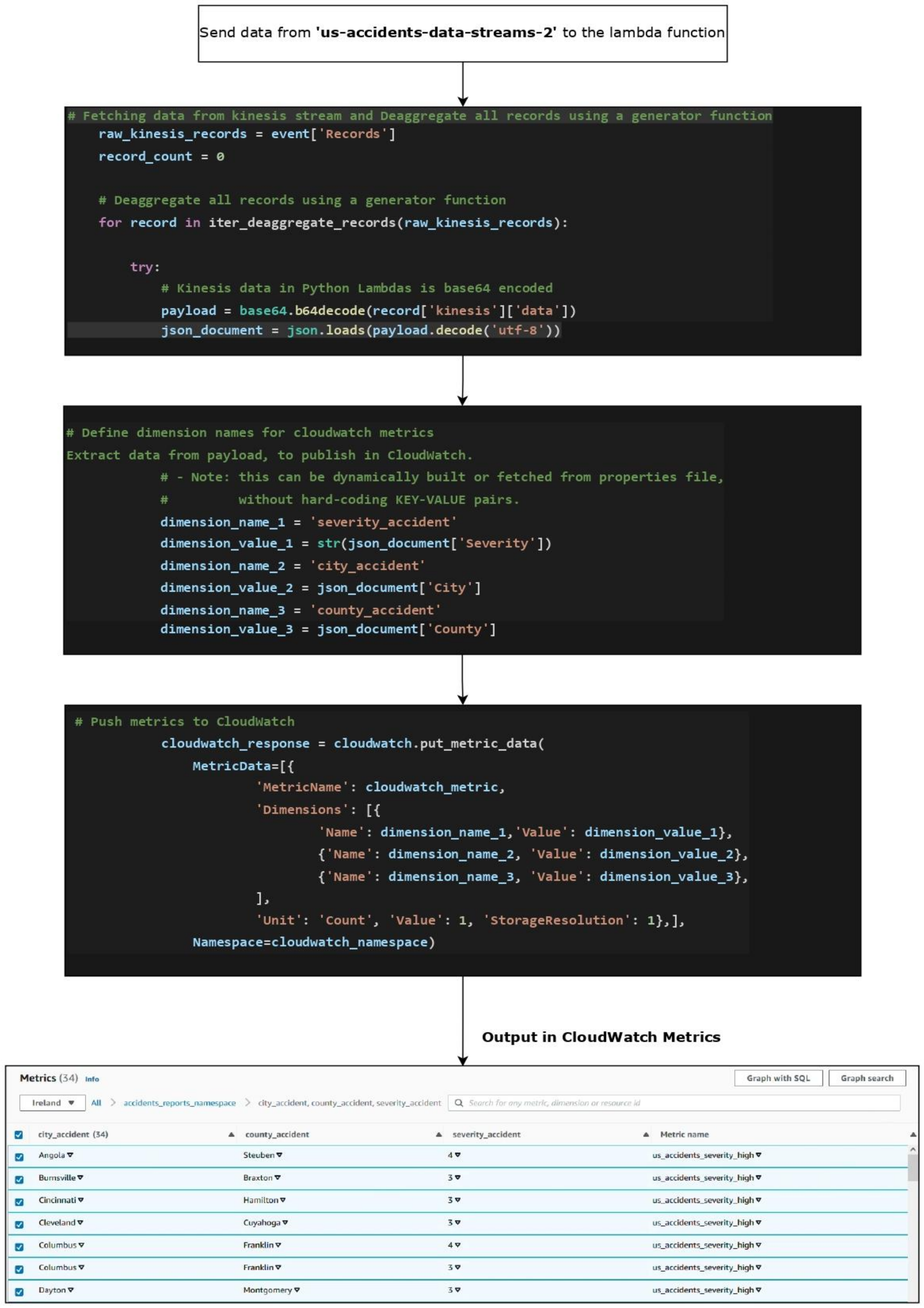
Real-Time Layer:



Steps in Real-Time Layer

1. Read the stream in Lambda and deaggregate the records using Kinesis Producer Library (KPL) (loop-in the generator to start receiving records)
    a. *pip install aws_kinesis_agg*
2. Decode the data in Lambda as the event record data is base64 encoded
3. Create CloudWatch metrics for 'Severity', 'City' and 'County
4. Push the metrics to CloudWatch with a Severity > 2
5. Create Grafana Dashboard for visualizing the data points
6. Setup Email Notifications through AWS SNS to manage severity > 4

Code Flow for Lambda



Send data from **'us-accidents-data-streams-2'** to the lambda function

```
# Fetching data from kinesis stream and Deaggregate all records using a generator function
    raw_kinesis_records = event['Records']
    record_count = 0

    # Deaggregate all records using a generator function
    for record in iter_deaggregate_records(raw_kinesis_records):

        try:
            # Kinesis data in Python Lambdas is base64 encoded
            payload = base64.b64decode(record['kinesis']['data'])
            json_document = json.loads(payload.decode('utf-8'))
```

```
# Define dimension names for cloudwatch metrics
Extract data from payload, to publish in CloudWatch.
            # - Note: this can be dynamically built or fetched from properties file,
            #         without hard-coding KEY-VALUE pairs.
            dimension_name_1 = 'severity_accident'
            dimension_value_1 = str(json_document['Severity'])
            dimension_name_2 = 'city_accident'
            dimension_value_2 = json_document['City']
            dimension_name_3 = 'county_accident'
            dimension_value_3 = json_document['County']
```

```
 # Push metrics to CloudWatch
            cloudwatch_response = cloudwatch.put_metric_data(
                MetricData=[{
                        'MetricName': cloudwatch_metric,
                        'Dimensions': [{
                                'Name': dimension_name_1,'Value': dimension_value_1},
                                {'Name': dimension_name_2, 'Value': dimension_value_2},
                                {'Name': dimension_name_3, 'Value': dimension_value_3},
                        ],
                        'Unit': 'Count', 'Value': 1, 'StorageResolution': 1},],
                Namespace=cloudwatch_namespace)
```

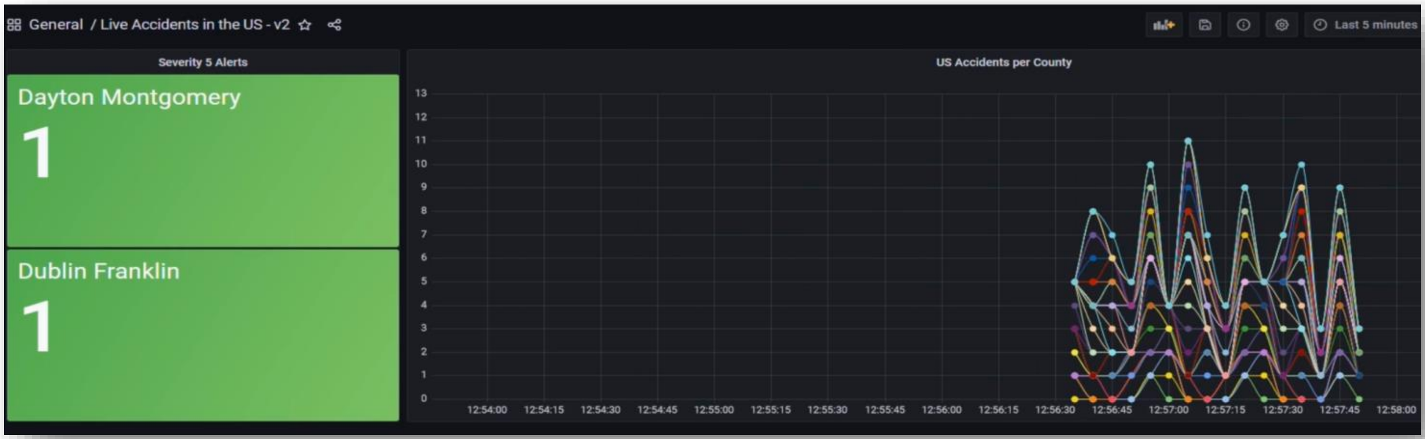
Output in CloudWatch Metrics

Metrics (34) Info

Ireland ▼ All > accidents_reports_namespace > city_accident, county_accident, severity_accident

city_accident (34)	county_accident	severity_accident	Metric name
Angola	Steuben	4	us_accidents_severity_high
Burnsville	Braxton	3	us_accidents_severity_high
Cincinnati	Hamilton	3	us_accidents_severity_high
Cleveland	Cuyahoga	3	us_accidents_severity_high
Columbus	Franklin	4	us_accidents_severity_high
Columbus	Franklin	3	us_accidents_severity_high
Dayton	Montgomery	3	us_accidents_severity_high

Grafana Dashboard:



Did you Know?
These [terminologies and concepts](#) are central to your understanding and use of Amazon CloudWatch.

Summary

- In this training, we simulated a real-time streaming environment for situations of emergence and how appropriate agencies can be alerted about them.
- The reusability of the proposed pipeline in the various industries was discussed.
- Initially, we discussed the concept of Perishable Data and how the value and use of data can degrade and change over time, respectively.
- Parallels between Batch and Stream Data types were drawn.
- We used US accident data for the demo purpose and defined three set industry standard pipelines for the incoming data stream.
- The architecture was divided into Raw-Layer, Analytical Layer, and Real-time Layer to best manage, analyze and take actions based on the accident's severity.
    - In the Raw-Layer, we changed the datatypes of the raw data, attached a 'Txn_Timestamp' to each data point, and pushed the stream to Kinesis.
    - In the Analytical layer, we transformed the data, filtering only cases with a severity>2 using Flink and storing them in a Glue Database.
    - Finally, we wrote a lambda function to receive the stream from Kinesis and create CloudWatch metrics, which were used for visualizing in Grafana.
    - The development of the Real-Time layer was concluded by creating a notification system for highly severe accidents.
- Various AWS services were leveraged to implement the concepts with the most cost-optimized configurations that can even serve 100x the amount of demo data used.