AWS Big Data Blog

Unified serverless streaming ETL architecture with Amazon Kinesis Data Analytics

by Ram Vittal and Akash Bhatia | on 30 SEP 2020 | in Amazon DynamoDB, Amazon Kinesis, Analytics, AWS Lambda, Kinesis Data Analytics, Kinesis Data Firehose, Kinesis Data Streams, Migration | Permalink | Comments | Share

Businesses across the world are seeing a massive influx of data at an enormous pace through multiple channels. With the advent of cloud computing, many companies are realizing the benefits of getting their data into the cloud to gain meaningful insights and save costs on data processing and storage. As businesses embark on their journey towards cloud solutions, they often come across challenges involving building serverless, streaming, real-time ETL (extract, transform, load) architecture that enables them to extract events from multiple streaming sources, correlate those streaming events, perform enrichments, run streaming analytics, and build data lakes from streaming events.

In this post, we discuss the concept of unified streaming ETL architecture using a generic serverless streaming architecture with Amazon Kinesis Data Analytics at the heart of the architecture for event correlation and enrichments. This solution can address a variety of streaming use cases with various input sources and output destinations. We then walk through a specific implementation of the generic serverless unified streaming architecture that you can deploy into your own AWS account for experimenting and evolving this architecture to address your business challenges.

Overview of solution

As data sources grow in volume, variety, and velocity, the management of data and event correlation become more challenging. Most of the challenges stem from data silos, in which different teams and applications manage data and events using their own tools and processes.

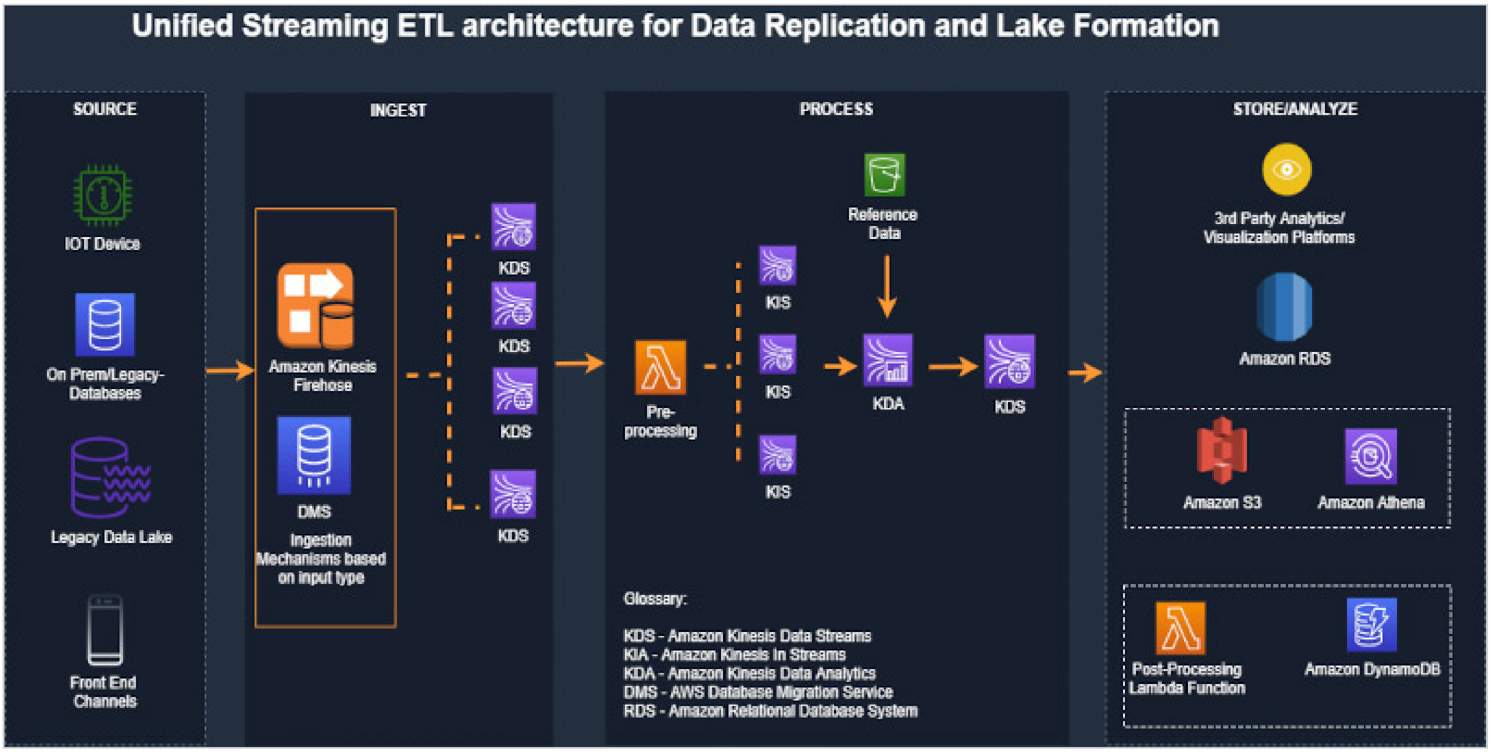
Modern businesses need a single, unified view of the data environment to get meaningful insights through streaming multi-joins, such as the correlation of sensory events and time-series data. Event correlation plays a vital role in automatically reducing noise and allowing the team to focus on those issues that really matter to the business objectives.

To realize this outcome, the solution proposes creating a three-stage architecture:

- Ingestion
- Processing
- Analysis and visualization

The source can be a varied set of inputs comprising structured datasets like databases or raw data feeds like sensor data that can be ingested as single or multiple parallel streams. The solution envisions multiple hybrid data sources as well. After it's ingested, the data is divided into single or multiple data streams depending on the use case and passed through a preprocessor (via an AWS Lambda function). This highly customizable processor transforms and cleanses data to be processed through analytics application. Furthermore, the architecture allows you to enrich data or validate it against standard sets of reference data, for example validating against postal codes for address data received from the source to verify its accuracy. After the data is processed, it's sent to various sink platforms depending on your preferences, which could range from storage solutions to visualization solutions, or even stored as a dataset in a high-performance database.

The solution is designed with flexibility as a key tenant to address multiple, real-world use cases. The following diagram illustrates the solution architecture.


Unified Streaming ETL architecture for Data Replication and Lake Formation
SOURCE
IOT Device
On Prem/Legacy-Databases
Legacy Data Lake
Front End Channels
INGEST
Amazon Kinesis Firehose
DMS
Ingestion Mechanisms based on input type
KDS
KDS
KDS
KDS
PROCESS
Pre-processing
KIS
KIS
KIS
Reference Data
KDA
KDS
Glossary:
KDS - Amazon Kinesis Data Streams
KIA - Amazon Kinesis In Streams
KDA - Amazon Kinesis Data Analytics
DMS - AWS Database Migration Service
RDS - Amazon Relational Database System
STORE/ANALYZE
3rd Party Analytics/ Visualization Platforms
Amazon RDS
Amazon S3
Amazon Athena
Post-Processing Lambda Function
Amazon DynamoDB

The architecture has the following workflow:

1. We use AWS Database Migration Service (AWS DMS) to push records from the data source into AWS in real time or batch.
2. AWS DMS writes records to Amazon Kinesis Data Streams. The data is split into multiple streams as necessitated through the channels.
3. A Lambda function picks up the data stream records and preprocesses them (adding the record type). This is an optional step, depending on your use case.
4. Processed records are sent to the Kinesis Data Analytics application for querying and correlating in-application streams, taking into account Amazon Simple Storage Service (Amazon S3) reference data for enrichment.

Solution walkthrough

For this post, we demonstrate an implementation of the unified streaming ETL architecture using Amazon RDS for MySQL as the data source and Amazon DynamoDB as the target. We use a simple order service data model that comprises orders, items, and products, where an order can have multiple items and the product is linked to an item in a reference relationship that provides detail about the item, such as description and price.

We implement a streaming serverless data pipeline that ingests orders and items as they are recorded in the source system into Kinesis Data Streams via AWS DMS. We build a Kinesis Data Analytics application that correlates orders and items along with reference product information and creates a unified and enriched record. Kinesis Data Analytics outputs output this unified and enriched data to Kinesis Data Streams. A Lambda function consumer processes the data stream and writes the unified and enriched data to DynamoDB.

To launch this solution in your AWS account, use the GitHub repo.

Prerequisites

Before you get started, make sure you have the following prerequisites:

- An AWS account
- Maven 3.x installed
- JDK 1.8+ installed
- The AWS Cloud Development Kit (AWS CDK)
- MySQL Workbench

Setting up AWS resources in your account

To set up your resources for this walkthrough, complete the following steps:

1. Set up the AWS CDK for Java on your local workstation. For instructions, see Getting Started with the AWS CDK.
2. Install Maven binaries for Java if you don't have Maven installed already.
3. If this is the first installation of the AWS CDK, make sure to run cdk bootstrap.
4. Clone the following GitHub repo.
5. Navigate to the project root folder and run the following commands to build and deploy:
    a. `mvn compile`
    b.
    ```
    cdk deploy UnifiedStreamETLCommonStack UnifiedStreamETLDataStack
    UnifiedStreamETLProcessStack
    ```

Setting up the orders data model for CDC

In this next step, you set up the orders data model for change data capture (CDC).

1. On the Amazon Relational Database Service (Amazon RDS) console, choose **Databases**.
2. Choose your database and make sure that you can connect to it securely for testing using bastion host or other mechanisms (not detailed in scope of this post).
3. Start MySQL Workbench and connect to your database using your DB endpoint and credentials.
4. To create the data model in your Amazon RDS for MySQL database, run orderdb-setup.sql.
5. On the AWS DMS console, test the connections to your source and target endpoints.
6. Choose **Database migration tasks**.
7. Choose your AWS DMS task and choose **Table statistics**.
8. To update your table statistics, restart the migration task (with full load) for replication.
9. From your MySQL Workbench session, run orders-data-setup.sql to create orders and items.
10. Verify that CDC is working by checking the **Table statistics**

Setting up your Kinesis Data Analytics application

To set up your Kinesis Data Analytics application, complete the following steps:

1. Upload the product reference products.json to your S3 bucket with the logical ID prefix `unifiedBucketId` (which was previously created by `cdk deploy`).

You can now create a Kinesis Data Analytics application and map the resources to the data fields.

2. On the Amazon Kinesis console, choose **Analytics Application**.
3. Choose **Create application**.
4. For **Runtime**, choose **SQL**.
5. Connect the streaming data created using the AWS CDK as a unified order stream.
6. Choose **Discover schema** and wait for it to discover the schema for the unified order stream. If discovery fails, update the records on the source Amazon RDS tables and send streaming CDC records.
7. Save and move to the next step.
8. Connect the reference S3 bucket you created with the AWS CDK and uploaded with the reference data.
9. Input the following:
    a. “products.json” on the path to the S3 object
    b. Products on the in-application reference table name
10. Discover the schema, then save and close.
11. Choose **SQL Editor** and start the Kinesis Data Analytics application.
12. Edit the schema for `SOURCE_SQL_STREAM_001` and map the data resources as follows:

Column Name	**Column Type**	**Row Path**
orderId	INTEGER	$.data.orderId
itemId	INTEGER	$.data.orderId
itemQuantity	INTEGER	$.data.itemQuantity
itemAmount	REAL	$.data.itemAmount
itemStatus	VARCHAR	$.data.itemStatus
COL_timestamp	VARCHAR	$.metadata.timestamp
recordType	VARCHAR	$.metadata.table-name
operation	VARCHAR	$.metadata.operation
partitionkeytype	VARCHAR	$.metadata.partition-key-type
schemaname	VARCHAR	$.metadata.schema-name
tablename	VARCHAR	$.metadata.table-name

transactionid	BIGINT	$.metadata.transaction-id
orderAmount	DOUBLE	$.data.orderAmount
orderStatus	VARCHAR	$.data.orderStatus
orderDateTime	TIMESTAMP	$.data.orderDateTime
shipToName	VARCHAR	$.data.shipToName
shipToAddress	VARCHAR	$.data.shipToAddress
shipToCity	VARCHAR	$.data.shipToCity
shipToState	VARCHAR	$.data.shipToState
shipToZip	VARCHAR	$.data.shipToZip

13. Choose **Save schema and update stream samples**.

When it's complete, verify for 1 minute that nothing is in the error stream. If an error occurs, check that you defined the schema correctly.

14. On your Kinesis Data Analytics application, choose your application and choose **Real-time analytics**.
15. Go to the SQL results and run kda-orders-setup.sql to create in-application streams.
16. From the application, choose **Connect to destination**.
17. For **Kinesis data stream**, choose **unifiedOrderEnrichedStream**.
18. For **In-application stream**, choose **ORDER_ITEM_ENRICHED_STREAM**.
19. Choose **Save** and **Continue**.

Testing the unified streaming ETL architecture

You're now ready to test your architecture.

1. Navigate to your Kinesis Data Analytics application.
2. Choose your app and choose **Real-time analytics**.
3. Go to the SQL results and choose **Real-time analytics**.
4. Choose the in-application stream `ORDER_ITEM_ENRCIHED_STREAM` to see the results of the real-time join of records from the order and order item streaming Kinesis events.
5. On the Lambda console, search for `UnifiedStreamETLProcess`.
6. Choose the function and choose **Monitoring**, **Recent invocations**.
7. Verify the Lambda function run results.
8. On the DynamoDB console, choose the `OrderEnriched` table.
9. Verify the unified and enriched records that combine order, item, and product records.

The following screenshot shows the `OrderEnriched` table.

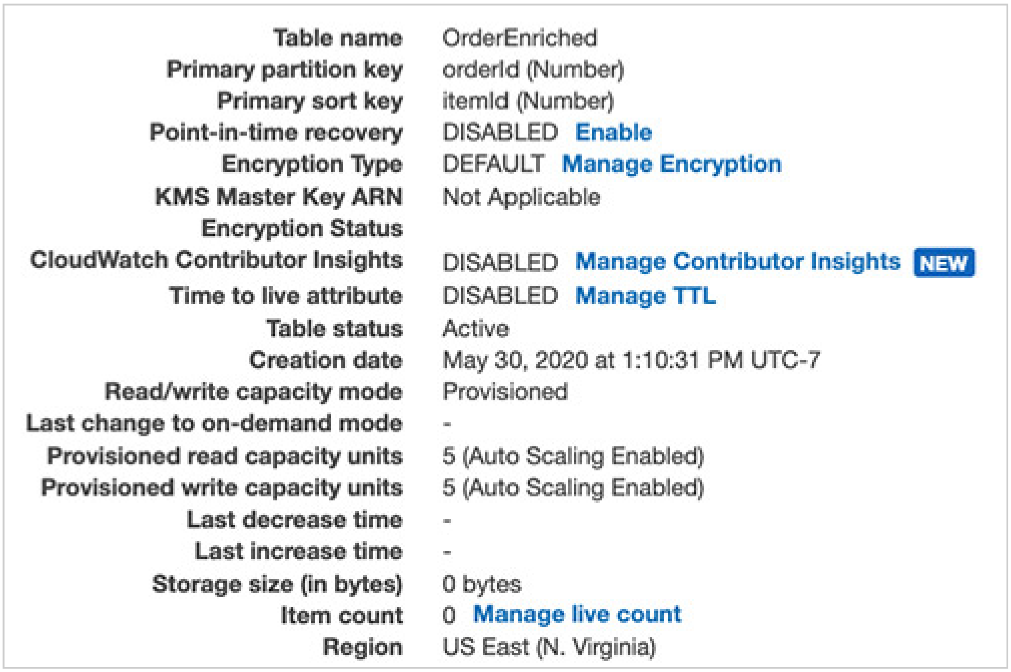


Table name	OrderEnriched
Primary partition key	orderId (Number)
Primary sort key	itemId (Number)
Point-in-time recovery	DISABLED Enable
Encryption Type	DEFAULT Manage Encryption
KMS Master Key ARN	Not Applicable
Encryption Status	
CloudWatch Contributor Insights	DISABLED Manage Contributor Insights NEW
Time to live attribute	DISABLED Manage TTL
Table status	Active
Creation date	May 30, 2020 at 1:10:31 PM UTC-7
Read/write capacity mode	Provisioned
Last change to on-demand mode	-
Provisioned read capacity units	5 (Auto Scaling Enabled)
Provisioned write capacity units	5 (Auto Scaling Enabled)
Last decrease time	-
Last increase time	-
Storage size (in bytes)	0 bytes
Item count	0 Manage live count
Region	US East (N. Virginia)

Operational aspects

When you're ready to operationalize this architecture for your workloads, you need to consider several aspects:

- Monitoring metrics for Kinesis Data Streams: GetRecords. `IteratorAgeMilliseconds` , `ReadProvisionedThroughputExceeded` , and `WriteProvisionedThroughputExceeded`
- Monitoring metrics available for the Lambda function, including but not limited to `Duration` , `IteratorAge` , E `rror count and success rate` (%), `Concurrent executions,` and `Throttles`
- Monitoring metrics for Kinesis Data Analytics (`millisBehindLatest`)
- Monitoring DynamoDB provisioned read and write capacity units
- Using the DynamoDB automatic scaling feature to automatically manage throughput

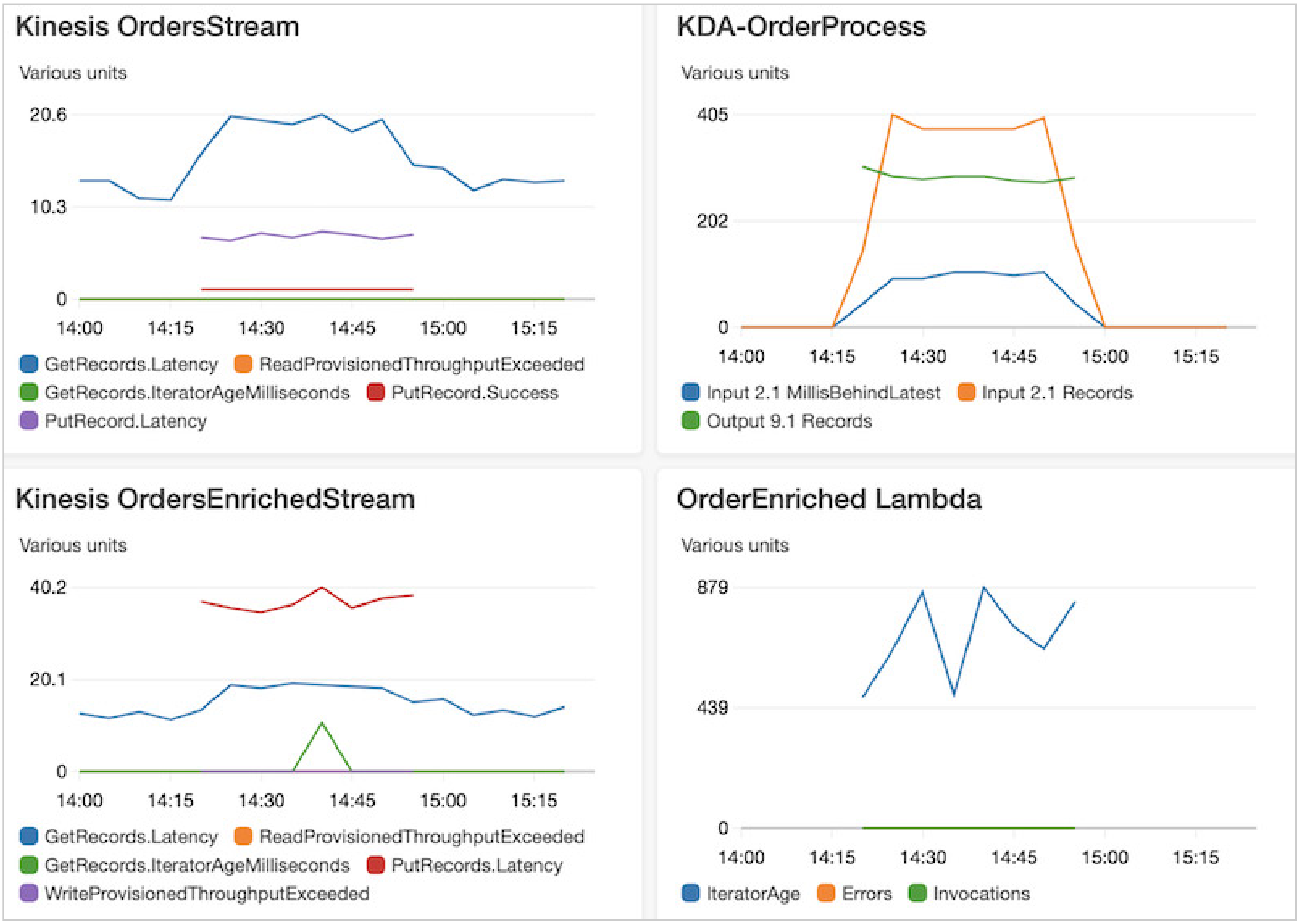
We used the solution architecture with the following configuration settings to evaluate the operational performance:

- Kinesis OrdersStream with two shards and Kinesis OrdersEnrichedStream with two shards
- The Lambda function code does asynchronous processing with Kinesis OrdersEnrichedStream records in concurrent batches of five, with batch size as 500
- DynamoDB provisioned WCU is 3000, RCU is 300

We observed the following results:

- 100,000 order items are enriched with order event data and product reference data and persisted to DynamoDB
- An average of 900 milliseconds latency from the time of event ingestion to the Kinesis pipeline to when the record landed in DynamoDB

The following screenshot shows the visualizations of these metrics.



Cleaning up

To avoid incurring future charges, delete the resources you created as part of this post (the AWS CDK provisioned AWS CloudFormation stacks).

Conclusion

In this post, we designed a unified streaming architecture that extracts events from multiple streaming sources, correlates and performs enrichments on events, and persists those events to destinations. We then reviewed a use case and walked through the code for ingesting, correlating, and consuming real-time streaming data with Amazon Kinesis, using Amazon RDS for MySQL as the source and DynamoDB as the target.

Managing an ETL pipeline through Kinesis Data Analytics provides a cost-effective unified solution to real-time and batch database migrations using common technical knowledge skills like SQL querying.

About the Authors

Ram Vittal is an enterprise solutions architect at AWS. His current focus is to help enterprise customers with their cloud adoption and optimization journey to improve their business outcomes. In his spare time, he enjoys tennis, photography, and movies.

Akash Bhatia is a Sr. solutions architect at AWS. His current focus is helping customers achieve their business outcomes through architecting and implementing innovative and resilient solutions at scale.