International Journal of Science and Research (IJSR)
ISSN: 2319-7064
Impact Factor 2024: 7.101

DynamoDB-Real Time Ingestion & Real Time Analytics

Khilawar Verma

Designation: Software Engineering Manager, SRE, Coinbase Inc, Santa Clara CA, USA

Abstract: ***This article offers a refreshing departure from traditional, cumbersome data ingestion practices by introducing a lean and real-time DynamoDB-based framework tailored to address long-standing operational bottlenecks in data pipelines. It is evident that existing ingestion models, often reliant on Change Data Capture (CDC) and file-based merges, have proven fragile, time-consuming, and riddled with maintenance challenges. This new framework, however, leverages AWS-native tools such as DynamoDB streams and Lambda functions to create a low-latency, highly resilient pipeline that simplifies cross-account data replication. What stands out is the dynamic schema evaluation, which means that developers no longer need to halt progress due to schema shifts-a common stumbling block in older systems. The framework's ability to maintain near-perfect data parity while providing robust monitoring and error recovery reflects a meaningful shift towards operational agility. This suggests that not only is data made available in near real-time, but it also empowers data engineers and analysts to move beyond verification and firefighting, focusing instead on deriving timely insights. Taken together, these innovations point to a practical yet powerful evolution in data ingestion, offering an almost plug-and-play solution that can be deployed swiftly without the typical overhead of complex ETL processes.***

Keywords: DynamoDB, AWS Public Cloud, Data Ingestion, BigData, Analytics, Data Pipelines

1. Overview

Today it takes a long time to move the data from the source database to the lake. Most of the current pipelines are batch oriented, complex processes and go through multiple frameworks and processing before the data is consumed. Data Analytics is T-1 day (In some scenarios, it could take up-to a week)

Patterns & Frameworks for Data Ingestion
CDC (Change Data Capture) + File Based Merge

Capture table-level inserts, updates, deletes (using OGG trail files), and apply these change events to a materialized hive view on a per event basis. KPIs/aggregates can be generated on the materialized view for reporting. The pattern is similar to capture table change events and merge into a materialized view. CDC + Merge is non-trivial to get right in production.
Data as Events (v4++)

Business events published on the Kafka bus consumed by platform microservices as well as reporting/analytics.

Replication-based
Companies such as AirBnB use database log shipping for ingestion into the Data Lake (specifically mysql binlog shipping to HBase). In AWS, DMS as a managed replication engine uses Replication model but DMS doesn't support DynamoDB based replication ingestion hence we have built our own DMS like solution to support DynamoDB ingestion framework which we are going to talk about here.

Some existing current Ingestion Patterns and it's Limitations
- Most of the ingestion patterns implement CDC + Merge pattern which is time consuming and batch oriented complex process with operational overhead and maintenance
- Challenges during mergers in S3
- There are many incidents and issues in these data ingestion pipelines. The existing pattern of database CDC (Change Data Capture) + file-based merge to create a materialized view is fragile and error-prone. Below pie chart depicts drill down of Data Incidents (per IOC Classification) impacting availability, accuracy or consumability of data.

Analysts and Scientists Pain-points
- Spend hours verifying dashboard results every week.
- The ETL logic has become layers and layers of band-aids.
- T-1 (24 hours) delayed insights and sometimes T-2 is not acceptable.
- Making a small fix or change in the pipeline can take weeks.

Data Engineer Pain-points
- Ingestion is fragile, changes in schema and increased load breaks the ingestion.
- Debugging is an unbounded nightmare along with Organizational Complexity
- Not possible to track analysts who wrote the existing ETL logic-- it's so complex, no one can explain.
- Ensuring timeliness working across cross BU teams is extremely complex with lots of to-&-fro.
- Extensibility of existing pipelines is extremely difficult.
- Need to build monitoring on entry and exit and keep on refining it.

Details/Benefits of the new pattern
As part of this framework we are using DynamoDB first. We are going to use AWS native DynamoDB replication technique
copy all source DynamoDB from various AWS accounts to one common Lake AWS account DynamoDB.

Pre-requisites

Below prerequisites are needed to set up the above framework.

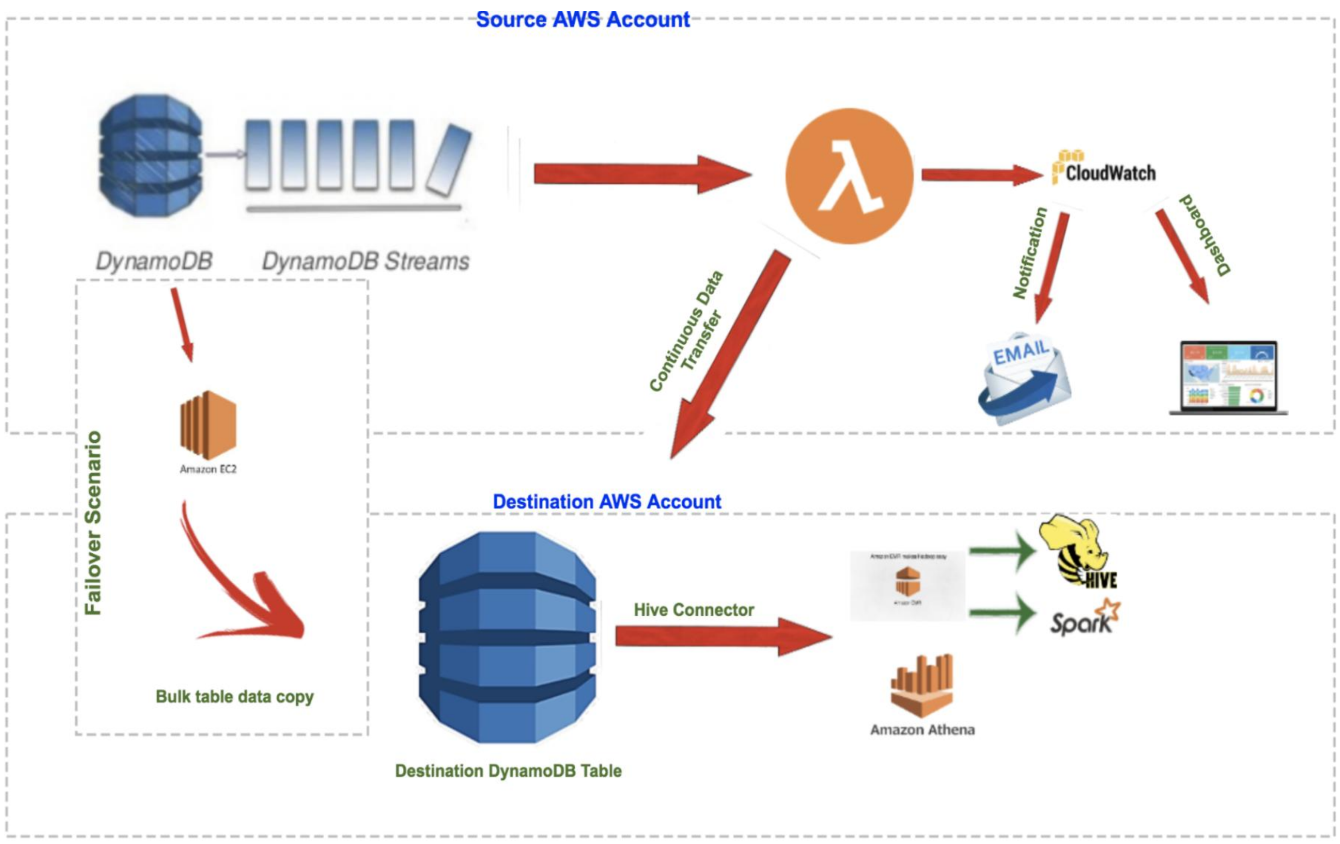
1) Either Source account access, or not then source account lead willing to set up.
2) Destination DynamoDB table needed.
3) New or existing EC2 is needed in the Source account if "Failure recovery option" is required.
4) Needs to enable DynamoDB stream in source account DynamoDB table.
5) needs to set up a lambda function in the source account.
6) Setup alerts in cloudwatch in the source account.
7) Lambda/cloudwatch role needs to be created in Source account.

Benefits

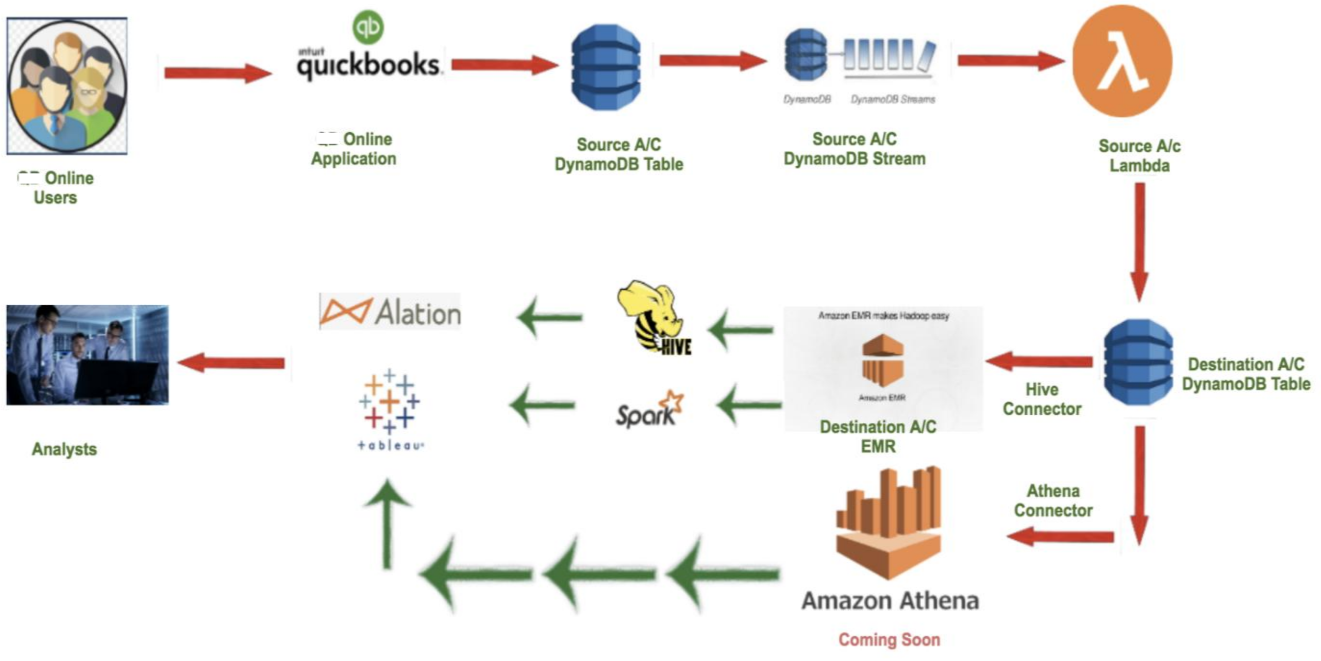
This technique has the following benefits.
1) It is the first ingestion framework for DynamoDB in Intuit.
2) It is real time framework (few micro seconds lag) vs t-1 to t-2 currently.
3) Dynamic Schema evaluation is incorporated in this framework, hence no Changes to ingestion pipeline in case of schema changes from dev side.
4) 0.0001% row parity difference between source and destination/lake tables (even though 1% row parity is acceptable)
5) Proper monitoring and alerting systems in place in case of any errors in ingestion pipeline.
6) Proper documentation and procedure created for Dev teams to create their own ingestion pipeline in self serve mode.
7) AWS supported replication techniques via DynamoDB stream.
8) AWS native tool lambda to run the replication logic
9) Failure recovery options available in case parity reaches more than 1%.
10) Very lightweight and takes only 15 min to set up the whole ingestion framework.

Architecture of the new Pattern

Below is the architecture diagram and flow chart that explains the flow of new ingestion framework.

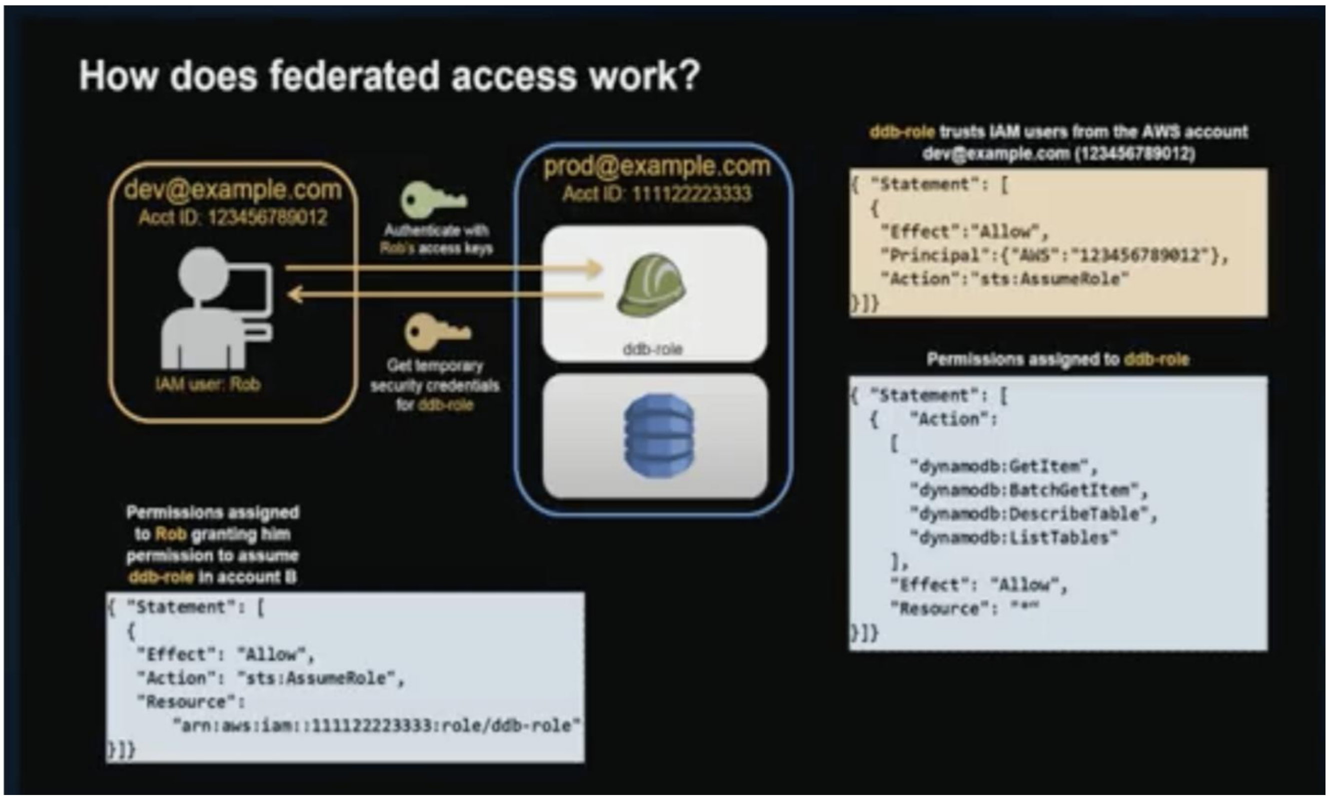


DynamoDb_Ingestion_Flow:



Total processing time of all 8 hops is within mili/micro seconds

Cross account access pattern
Below is the architecture diagram which shows how cross account DynamoDB ingestion will work.



How does federated access work?

ddb-role trusts IAM users from the AWS account dev@example.com (123456789012)

```
{ "Statement": [
  {
   "Effect":"Allow",
   "Principal":{"AWS":"123456789012"},
   "Action":"sts:AssumeRole"
}]}
```

Permissions assigned to ddb-role

```
{ "Statement": [
  {   "Action":
      [
       "dynamodb:GetItem",
       "dynamodb:BatchGetItem",
       "dynamodb:DescribeTable",
       "dynamodb:ListTables"
      ],
      "Effect": "Allow",
      "Resource": "*"
}]}
```

Permissions assigned to Rob granting him permission to assume ddb-role in account B

```
{ "Statement": [
  {
   "Effect": "Allow",
   "Action": "sts:AssumeRole",
   "Resource":
        "arn:aws:iam::111122223333:role/ddb-role"
}]}
```

How to Analyze the data

1) Through EMR hive
   We can access ingested data directly from Hive as it's connector supports DynamoDB access.
2) Through other tools like Alation/Tableau etc.

DynamoDB Limitations

There are some limitations in the DynamoDB as shown below

- We have only 10000 read or write units. We can increase them as well with special requests to AWS. We are currently using only 100 RSUs/WSUs in the destination Table.
- DynamoDB table names are unique hence we have to make sure that if we have similar names in some scenario, we will need to create tables with different names.
- You can have only 256 tables per region per account.

References

[1] Stonebraker, M., Çetintemel, U., & Zdonik, S. (2005). The 8 Requirements of Real-Time Stream Processing. *ACM SIGMOD Record*, 34 (4), 42–47.

[2] Gulisano, V., Jiménez-Peris, R., Patino-Martinez, M., Soriente, C., & Valduriez, P. (2012). StreamCloud: An Elastic and Scalable Data Streaming System. *IEEE Transactions on Parallel and Distributed Systems*, 23 (12), 2351–2365.

[3] Kreps, J., Narkhede, N., & Rao, J. (2011). Kafka: A Distributed Messaging System for Log Processing. *Proceedings of the NetDB Workshop*, 1–7.

[4] Abadi, D. J., Carney, D., Çetintemel, U., Cherniack, M., Convey, C., Lee, S., & Stonebraker, M. (2003). Aurora: A New Model and Architecture for Data Stream Management. *The VLDB Journal*, 12 (2), 120–139.

[5] Hirzel, M., Schneider, S., Gedik, B., & Grimm, R. (2014). Partition and Compose: Parallel Complex Event Processing. *Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems (DEBS)*, 125–136.