New Features in Amazon DynamoDB – PartiQL, Export to S3, Integration with Kinesis Data Streams

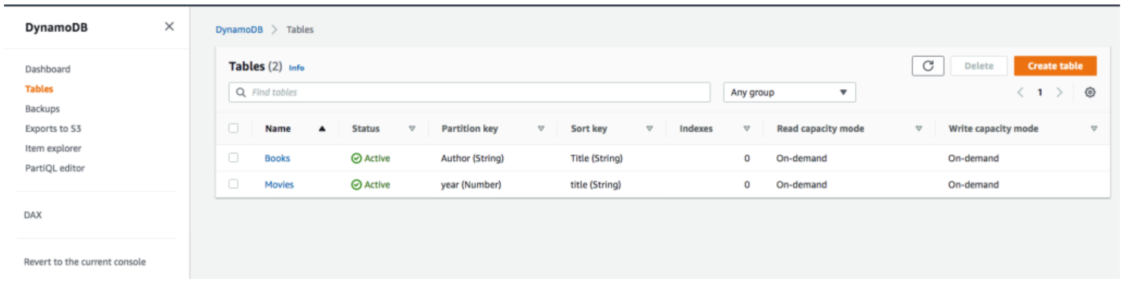
Posted on **December 14, 2020December 14, 2020** by **Anand**

Every time with AWS re:Invent around, AWS releases many new features over a period of month. In this blog post I will touch on 3 new features which were introduced for Amazon DynamoDB. DynamoDB is a non-relational managed database with single digit millisecond performance at any scale.

New Features in Amazon DynamoDB –

1. PartiQL – SQL-compatible query language for Amazon DynamoDB.
2. Export to S3 – Export Amazon DynamoDB table to S3. In this blog I have added a use-case of deserializing the DynamoDB items, writing it to S3 and query using Athena.
3. Direct integration of DynamoDB with Kinesis Streams – Stream item-level images of Amazon DynamoDB as a Kinesis Data Stream.

To start with, lets look at the new Amazon DynamoDB console. I have 2 DDB tables to play around with for this blog. The Books table is partitioned by Author and sorted by Title. The Movies table is partitioned by year and sorted by title.



Name	Status	Partition key	Sort key	Indexes	Read capacity mode	Write capacity mode
Books	Active	Author (String)	Title (String)	0	On-demand	On-demand
Movies	Active	year (Number)	title (String)	0	On-demand	On-demand

(https://aprakash.files.wordpress.com/2020/11/screen-shot-2020-11-23-at-9.18.58-pm.png)

Lets jump on to the features:

1. **PartiQL** – You can use SQL to select, insert, update and delete items from Amazon DynamoDB. Currently you can use PartiQL for DynamoDB from the Amazon DynamoDB console, the AWS Command Line Interface (AWS CLI), and DynamoDB APIs. For this blog, I am using the AWS console.

DynamoDB > PartiQL editor

SELECT SQLs –

Simple select SQL

```
SELECT * FROM Books where Author='William Shakespeare'
```

Title	Formats	Author	Category
Hamlet	{ “Hardcover” : { “S” : “GVJZQ7JK” }, “Paperback” : { “S” : “A4TFUR98” }, “Audiobook” : { “S” : “XWMGHW96” } }	William Shakespeare	Drama

The following SQL returns the title, hardcover and category using key path –

```
SELECT Title, Formats['Hardcover'], category FROM Books where
Author='John Grisham'
```

Category	Title	Hardcover
Suspense	The Firm	Q7QWE3U2
Suspense	The Rainmaker	J4SUKVGU
Thriller	The Reckoning	null

The following SQL uses “contains” function which returns TRUE if attribute category has string ‘Suspense’ –

```
SELECT Title, Formats['Audiobook'], Category FROM Books where
Author='John Grisham' and contains(Category, 'Suspense')
```

year	**title**	**release_date**	**rank**
2011	Sherlock Holmes: A Game of Shadows	2011-12-10T00:00:00Z	570

INSERT SQL –

REPORT THIS AD

Insert a single item –

```
 INSERT INTO Books value {'Title' : 'A time to kill', 'Author' : 'John
Grisham', 'Category' : 'Suspense' }

SELECT * FROM Books WHERE Title='A time to kill'
```

Author	Title	Category
John Grisham	A time to kill	Suspense

"INSERT INTO SELECT" SQL fails with ValidationException: Unsupported operation: Inserting multiple items in a single statement is not supported, use "INSERT INTO tableName VALUE item" instead

UPDATE SQL –

In the previous insert sql, Formats column was null. So lets update the Formats column for the book.

```
 UPDATE Books SET Formats={'Hardcover':'J4SUKVGU' ,'Paperback':
'D7YF4FCX'} WHERE Author='John Grisham' and Title='A time to kill'
```

Title	Formats	Author	Category
A time to kill	{"Hardcover":{"S":"J4SUKVGU"},"Paperback": {"S":"D7YF4FCX"}}	John Grisham	Suspense

You can use update sql to remove key from map –

```
UPDATE Books REMOVE Formats.Paperback WHERE Author='John Grisham' and
Title='A time to kill'
```

Title	Formats	Author	Category
A time to kill	{"Hardcover":{"S":"J4SUKVGU"}}	John Grisham	Suspense

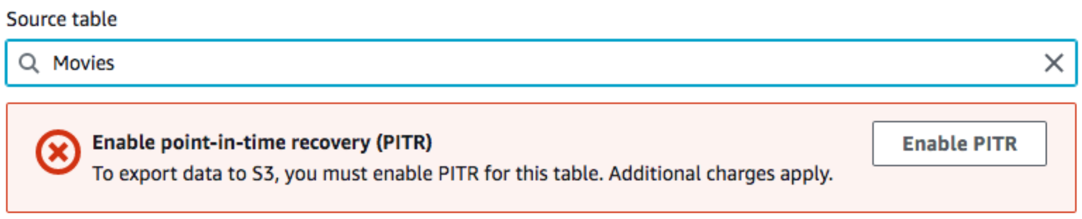
DELETE SQL –

```
DELETE FROM Books WHERE Author='John Grisham' and Title='A time to
kill'
```

For more references – **https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-reference.html (https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-reference.html)**

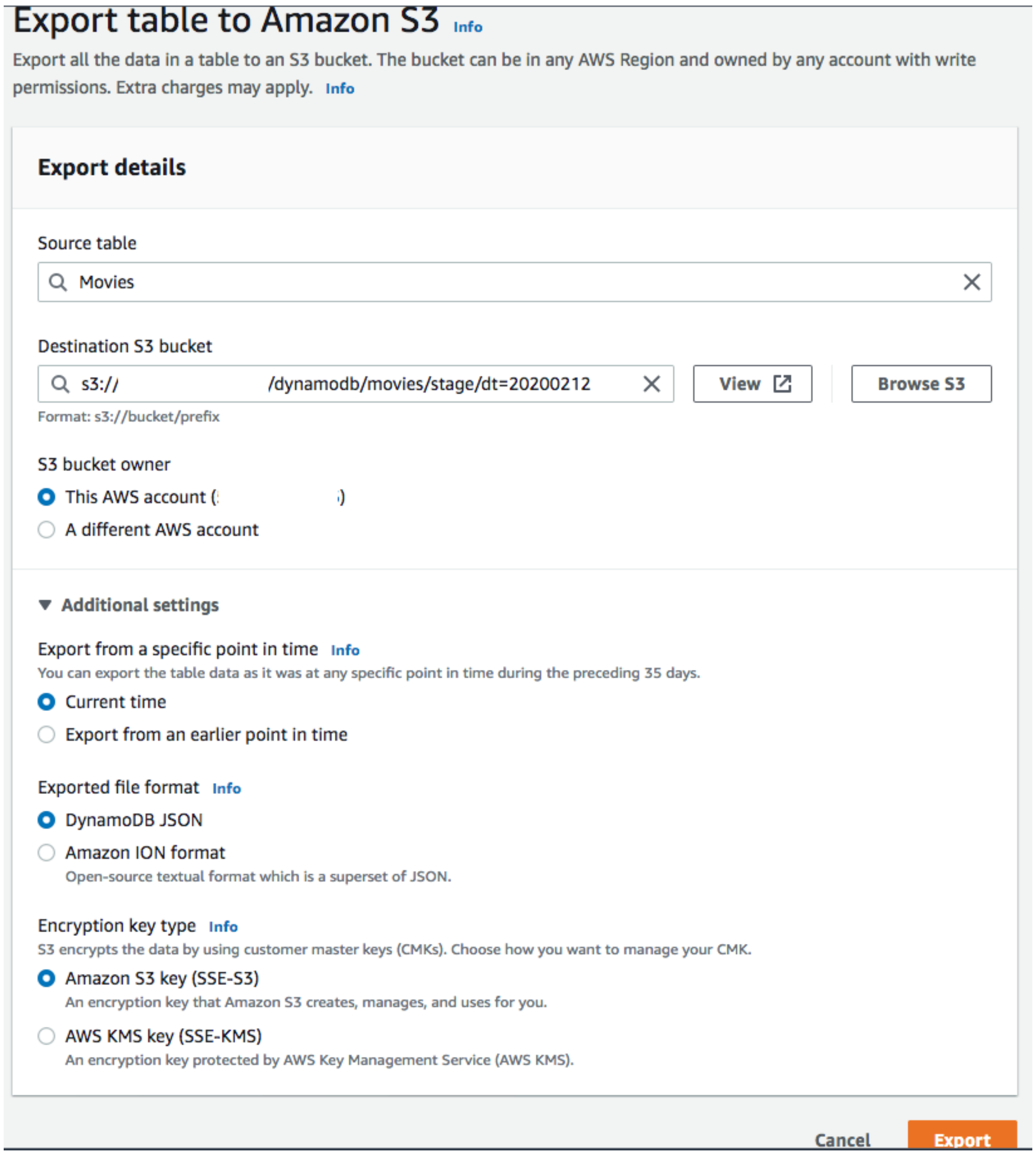
2. **EXPORT TO S3** – Export full Amazon DynamoDB table to Amazon S3 bucket. Using DynamoDB table export, you can export data from an Amazon DynamoDB table from any time within your point-in-time recovery window. To do so the table must have point in time recovery (PITR) enabled. If PITR is not enabled for the table, the Export to S3 will report error asking it to be enabled.

DynamoDB > Exports to S3



Enable point-in-time recovery (PITR)
To export data to S3, you must enable PITR for this table. Additional charges apply.

(https://aprakash.files.wordpress.com/2020/11/screen-shot-2020-11-23-at-11.03.16-pm.png)



Export table to Amazon S3 Info

Export all the data in a table to an S3 bucket. The bucket can be in any AWS Region and owned by any account with write permissions. Extra charges may apply. Info

Export details

Source table: Movies

Destination S3 bucket: s3:// /dynamodb/movies/stage/dt=20200212

Format: s3://bucket/prefix

S3 bucket owner
- [✓ This AWS account ( )]
- A different AWS account

Additional settings

Export from a specific point in time Info

You can export the table data as it was at any specific point in time during the preceding 35 days.
- [✓ Current time]
- Export from an earlier point in time

Exported file format Info
- [✓ DynamoDB JSON]
- Amazon ION format
  Open-source textual format which is a superset of JSON.

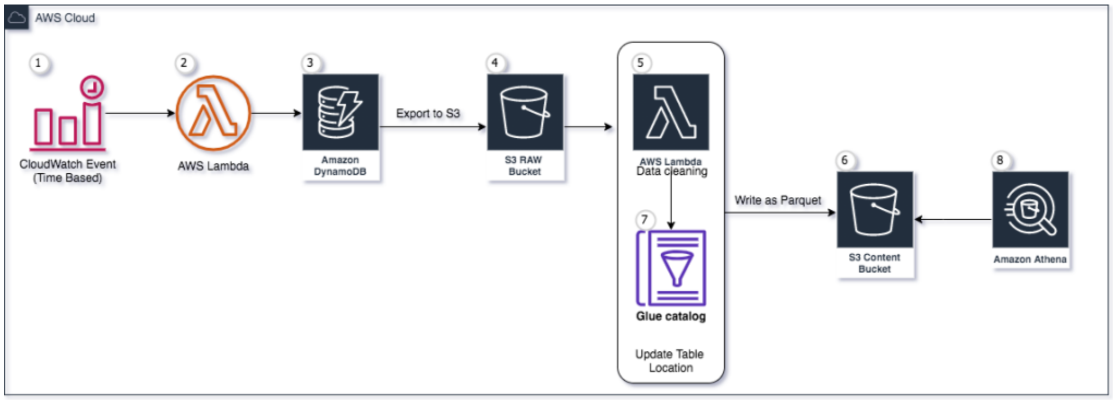
Encryption key type Info

S3 encrypts the data by using customer master keys (CMKs). Choose how you want to manage your CMK.
- [✓ Amazon S3 key (SSE-S3)]
  An encryption key that Amazon S3 creates, manages, and uses for you.
- AWS KMS key (SSE-KMS)
  An encryption key protected by AWS Key Management Service (AWS KMS).

(https://aprakash.files.wordpress.com/2020/12/screen-shot-2020-12-02-at-11.46.36-am-1.png)

After the export is complete it generates a manifest-summary.json file summarizing the export details and a manifest-file.json containing the details of S3 file locations.

Further to extend on Export to S3 feature to perform analytics and complex queries on your data you can create a data pipeline to deserialize the DynamoDB JSON format and query data from AWS Athena.

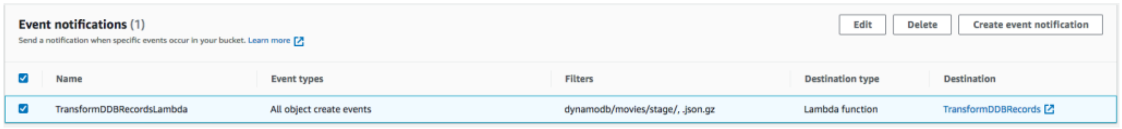


[(https://aprakash.files.wordpress.com/2020/12/ddb_pipeline_1-6.png)](https://aprakash.files.wordpress.com/2020/12/ddb_pipeline_1-6.png)

The workflow contains the following steps:

- Time based CloudWatch event is triggered.
- This event triggers AWS Lambda function.
- DynamoDB Export to S3 is initiated.
- It writes the data in DynamoDB JSON format to S3 RAW bucket. The S3 objects are zipped json files.
- For each .json.gz file in S3 RAW bucket an event notification is set to trigger AWS Lambda. The detail is show below in S3 Event to trigger AWS Lambda section.
- AWS Lambda reads the S3 object and deserializes the DynamoDB JSON format data using DynamoDB TypeDeserializer class. This class deserializes DynamoDB types to Python types. The deserialized data is written to S3 Content bucket in Parquet format. Code is in Lambda function code section.
- AWS Lambda function updates the table location in AWS Glue catalog.
- Query the data using AWS Athena.

S3 Event to trigger AWS Lambda



Event notifications (1)

Send a notification when specific events occur in your bucket. Learn more

Edit | Delete | Create event notification

	Name	Event types	Filters	Destination type	Destination
☑	TransformDDBRecordsLambda	All object create events	dynamodb/movies/stage/, .json.gz	Lambda function	TransformDDBRecords

[(https://aprakash.files.wordpress.com/2020/12/screen-shot-2020-12-02-at-11.48.08-am.png)](https://aprakash.files.wordpress.com/2020/12/screen-shot-2020-12-02-at-11.48.08-am.png)

Lambda function code

```
import io
import gzip
import json
import boto3
import uuid
import pandas as pd
import awswrangler as wr
from datetime import datetime
from urllib.parse import unquote_plus


def update_glue_table(*, database, table_name, new_location, region_nam
    """ Update AWS Glue non-partitioned table location
    """

    glue = boto3.client("glue", region_name=region_name)

    response = glue.get_table(
        DatabaseName=database, Name=table_name)

    table_input = response["Table"]
    current_location = table_input["StorageDescriptor"]["Location"]

    table_input.pop("UpdateTime", None)
    table_input.pop("CreatedBy", None)
    table_input.pop("CreateTime", None)
    table_input.pop("DatabaseName", None)
    table_input.pop("IsRegisteredWithLakeFormation", None)
    table_input.pop("CatalogId", None)

    table_input["StorageDescriptor"]["Location"] = new_location

    response = glue.update_table(
        DatabaseName=database,
        TableInput=table_input
    )

    return response


def lambda_handler(event, context):

    """
    Uses class TypeDeserializer which deserializes DynamoDB types to Py

    Example -

    raw data format :
        [{'ACTIVE': {'BOOL': True}, 'params': {'M': {'customer': {'S':
    deserialized data format:
        [{'ACTIVE': True, 'params': {'customer': 'TEST', 'index': Decim

    """


    s3client = boto3.client('s3')
    athena_db = "default"
    athena_table = "movies"

    for record in event['Records']:
        bucket = record['s3']['bucket']['name']
        key = unquote_plus(record['s3']['object']['key'])

        response = s3client.get_object(Bucket=bucket, Key=key)
        content = response['Body'].read()
        with gzip.GzipFile(fileobj=io.BytesIO(content), mode='rb') as f
            data = [json.loads(line) for line in fh]

    all_data = []

    boto3.resource('dynamodb')
    deserializer = boto3.dynamodb.types.TypeDeserializer()
    for row in data:
        all_data.append({k: deserializer.deserialize(v) for k,v in row[


    data_df = pd.DataFrame(all_data)

    dt = datetime.utcnow().strftime("%Y-%m-%d-%H-%M")
    s3_path="s3://%s/dynamodb/%s/content/dt=%s/" % (bucket, athena_tabl

    wr.s3.to_parquet(
        df=data_df,
        path=s3_path,
        dataset = True,
    )


    update_response = update_glue_table(
        database=athena_db,
        table_name=athena_table,
        new_location=s3_path,
        region_name="us-west-2")

    if update_response["ResponseMetadata"]["HTTPStatusCode"] == 200:
        return (f"Successfully updated glue table location - {athena_db
    else:
        return (f"Failed updating glue table location - {athena_db}.{at
```

REPORT THIS AD

Query data from Athena

```
SELECT title, info.actors, info.rating, info.release_date, year FROM mov
```

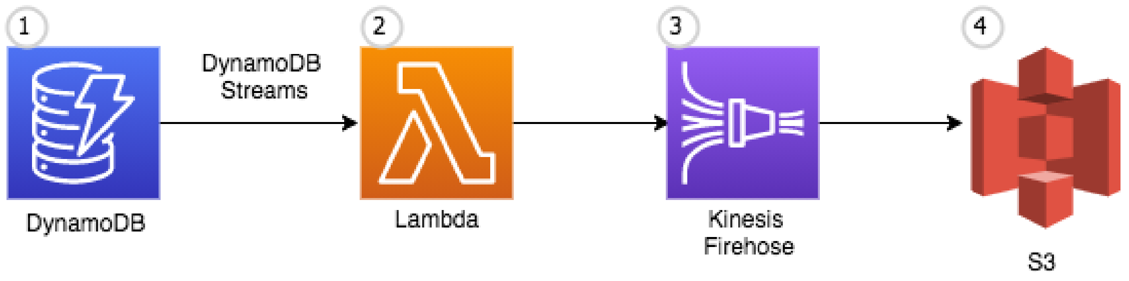
	title	actors	rating	release_date	year
1	Christmas Vacation	[Chevy Chase, Beverly D'Angelo, Juliette Lewis]	0.73	1989-11-30T00:00:00Z	1989

For more references –
https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/DataExport.html (https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/DataExport.html)

3. DynamoDB integration with Kinesis Stream.

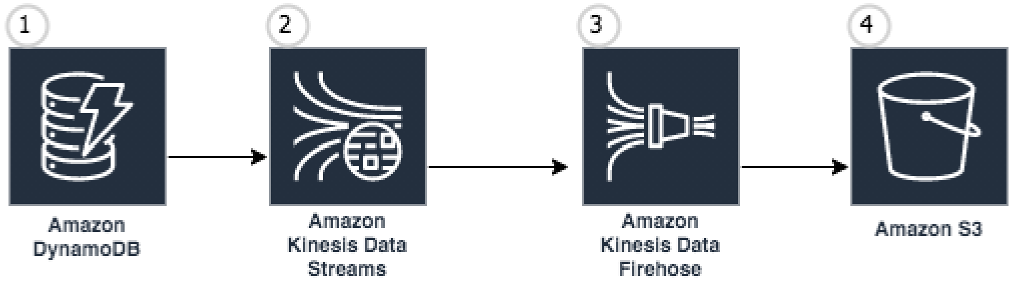
DynamoDB Streams captures a time-ordered sequence of item-level modifications in a DynamoDB table. Earlier to publish DynamoDB data to S3 in near real time, one of the ways was to enable DynamoDB streams and use AWS Lambda function to forward data to Kinesis Firehose which published the data to

S3. To do so you can use a handy package provided by aws labs – **https://github.com/awslabs/lambda-streams-to-firehose (https://github.com/awslabs/lambda-streams-to-firehose)** .



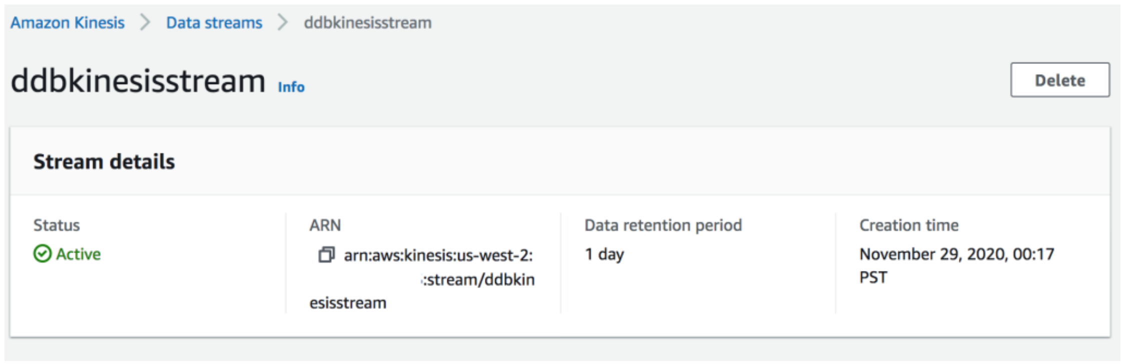
(https://aprakash.files.wordpress.com/2020/12/ddb_pipeline_1-1.png)

With several such use cases, AWS has now integrated AWS DynamoDB directly with Amazon Kinesis Stream. Now you can capture item-level changes in your DynamoDB tables as a Kinesis data stream. This can enable you to publish the data to S3 as shown in the below pipeline.



(https://aprakash.files.wordpress.com/2020/12/ddb_pipeline_2.png)

To setup the pipeline, create a Amazon Kinesis Data Stream.



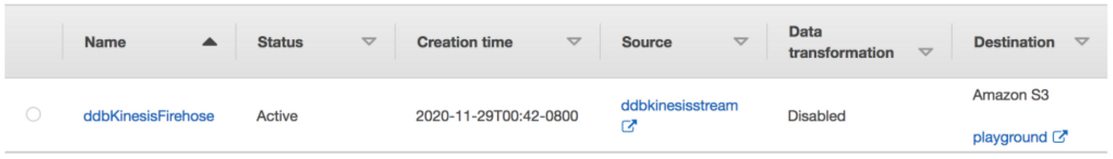
ddbkinesisstream

Stream details

Status	ARN	Data retention period	Creation time
Active	arn:aws:kinesis:us-west-2: :stream/ddbkinesisstream	1 day	November 29, 2020, 00:17 PST

(https://aprakash.files.wordpress.com/2020/12/screen-shot-2020-12-12-at-12.51.12-pm.png)

After setting up the Amazon Kinesis Data Stream, create Amazon Kinesis Firehose. The source to Kinesis Firehose will be Amazon Kinesis Data Stream and the destination will be S3.



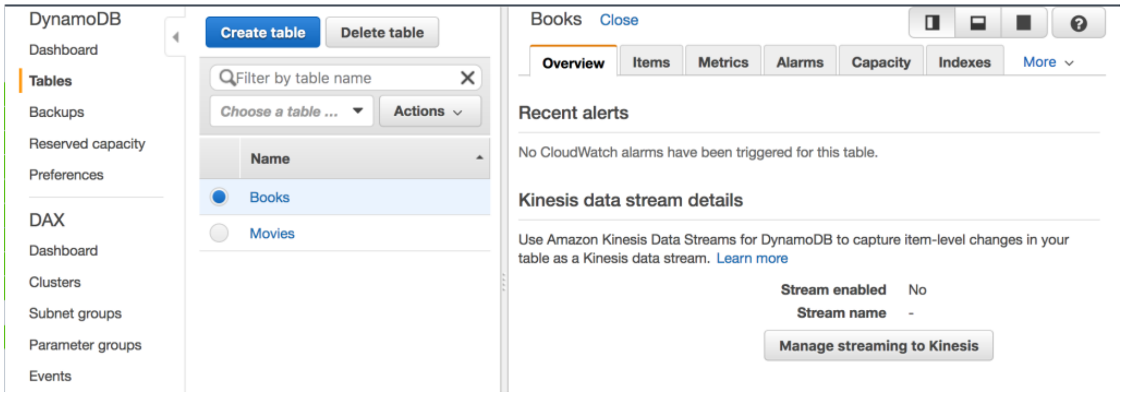
Name	Status	Creation time	Source	Data transformation	Destination
ddbKinesisFirehose	Active	2020-11-29T00:42-0800	ddbkinesisstream	Disabled	Amazon S3 playground

(https://aprakash.files.wordpress.com/2020/12/screen-shot-2020-12-12-at-12.54.37-pm.png)

To proceed move on to enable streaming to Kinesis.

REPORT THIS AD

DynamoDB > Table > Kinesis data stream details > Manage streaming to Kinesis



Recent alerts

No CloudWatch alarms have been triggered for this table.

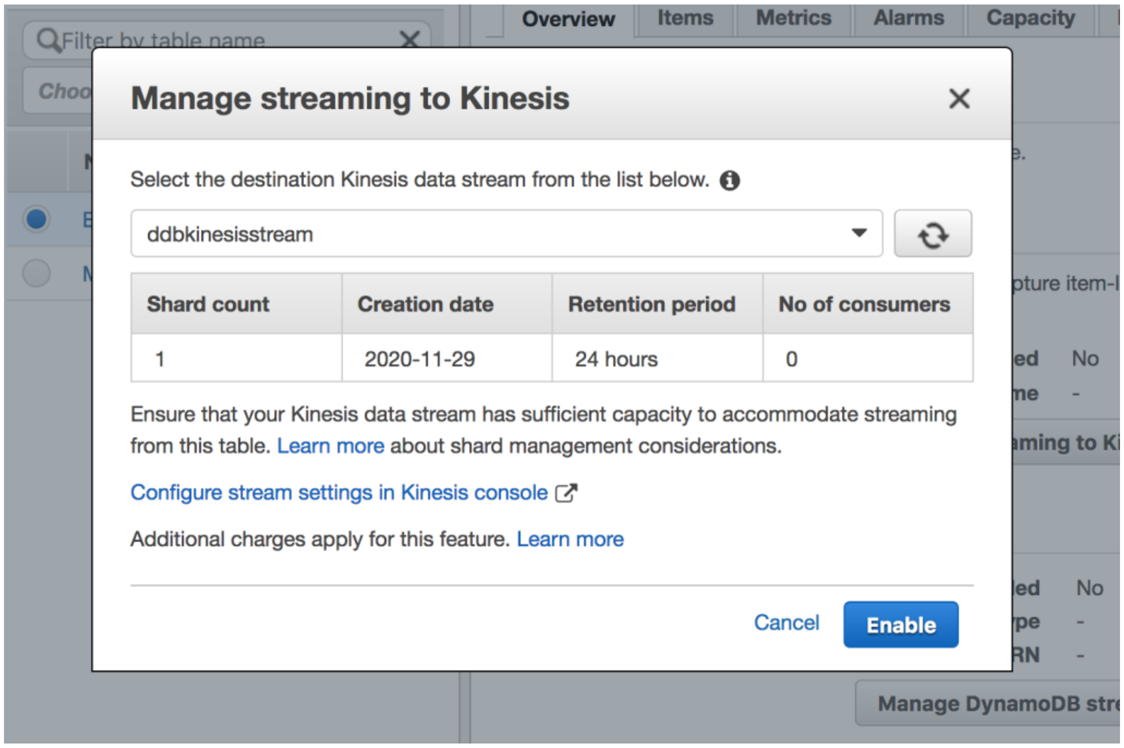
Kinesis data stream details

Use Amazon Kinesis Data Streams for DynamoDB to capture item-level changes in your table as a Kinesis data stream. Learn more

Stream enabled No
Stream name -

Manage streaming to Kinesis

(https://aprakash.files.wordpress.com/2020/12/screen-shot-2020-11-29-at-12.16.49-am.png)



Manage streaming to Kinesis

Select the destination Kinesis data stream from the list below.

ddbkinesisstream

Shard count	Creation date	Retention period	No of consumers
1	2020-11-29	24 hours	0

Ensure that your Kinesis data stream has sufficient capacity to accommodate streaming from this table. Learn more about shard management considerations.

Configure stream settings in Kinesis console

Additional charges apply for this feature. Learn more

(https://aprakash.files.wordpress.com/2020/12/screen-shot-2020-11-29-at-12.43.04-am.png)

Once the stream is enabled any item-level change in the table will be captured and written to Amazon S3 bucket. Below is an example of the record which was updated in DynamoDB using PartiQL. The record contains approximate creation date time of the record in DynamoDB streams, along with New and Old image of the record. These records can be parsed using AWS Lambda or AWS Glue and stored in Data Lake for analytical use-cases.

```
{
   "awsRegion": "us-west-2",
   "dynamodb": {
     "ApproximateCreationDateTime": 1606714671542,
     "Keys": {
       "Author": {
         "S": "James Patterson"
       },
       "Title": {
         "S": "The President Is Missing"
       }
     },
     "NewImage": {
       "Title": {
         "S": "The President Is Missing"
       },
       "Formats": {
         "M": {
           "Hardcover": {
             "S": "JSU4KGVU"
           }
         }
       },
       "Author": {
         "S": "James Patterson"
       },
       "Category": {
         "S": "Mystery"
       }
     },
     "OldImage": {
       "Title": {
         "S": "The President Is Missing"
       },
       "Formats": {
         "M": {
           "Hardcover": {
             "S": "JSU4KGVU"
           },
           "Paperback": {
             "S": "DY7F4CFX"
           }
         }
       },
       "Author": {
         "S": "James Patterson"
       },
      "Category": {
        "S": "Mystery"
      }
    },
    "SizeBytes": 254
  },
  "eventID": "bcaaf073-7e0d-49c2-818e-fe3cf7e5f18a",
  "eventName": "MODIFY",
  "userIdentity": null,
  "recordFormat": "application/json",
  "tableName": "Books",
  "eventSource": "aws:dynamodb"
}
```

REPORT THIS AD

To conclude, in this post I have introduced you to PartiQL which provides SQL-compatible query for Amazon DynamoDB. We also looked at the Export to S3 feature and how we can create an end to end pipeline to query the Amazon DynamoDB data in Amazon S3 bucket using AWS Athena. And finally we looked at real time analytics use-case where you can enable streams to capture item-level changes in Amazon DynamoDB table as Kinesis data streams.

Advertisements

REPORT THIS AD

Posted in **AWS** **Leave a comment**

Website Powered by WordPress.com.