Choose a Topic

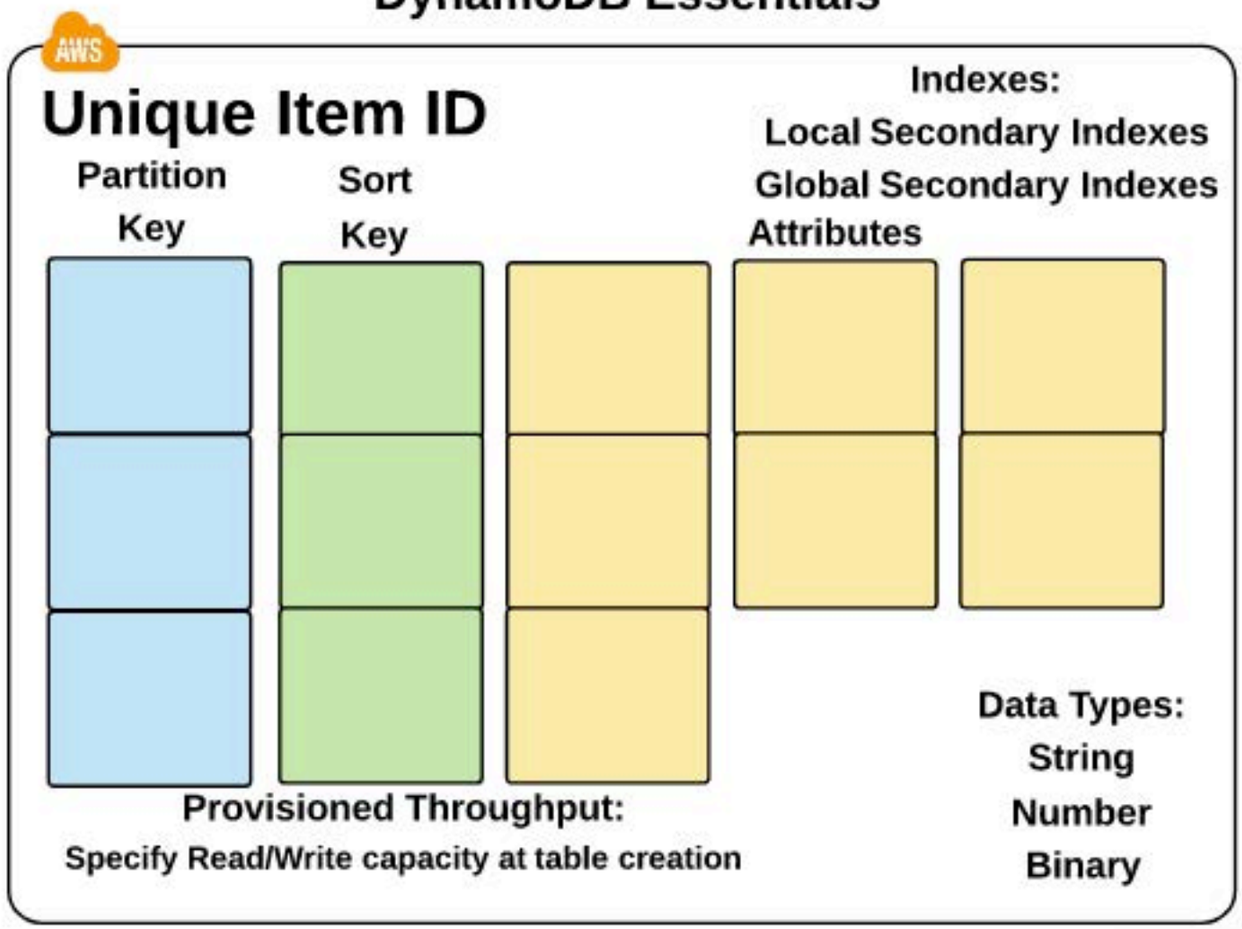
- Introduction
- Auto Scaling Deployment Concepts
- Deployment Concepts With EC2
- CloudWatch For DevOps
- CloudFormation For DevOps
- Elastic Beanstalk For DevOps
- Application Deployments With OpsWorks
- DynamoDB Concepts
- S3 Concepts For DevOps
- Blue/Green Deployments
- Scenario Solver
- Deployment Pipelines
- API Gateway
- Lambda

Using DynamoDB with Applications

Scenario: Amazon DynamoDB is a fully managed NoSQL database service that makes it simple and cost-effective to store and retrieve any amount of data and serve any level of request traffic. It is important to understand the foundational concepts of DynamoDB, such as Partition Keys and Sort Keys as well as Local and Global Secondary indexes. An understanding of Read and Write capacity units is also important. In addition, at the DevOps Pro level it is important to understand the use case for DynamoDB, such as storing metadata and the services used to export data to DnyamoDB (Data Pipline, DynamoDB Streams.

- Secondary Indexes
- Provisioned Throughput
- In-memory Acceleration - Dax
- Web Identity Federation
- Data Pipeline
- DynamoDB Streams

DynamoDB Essentials



Choose a Topic

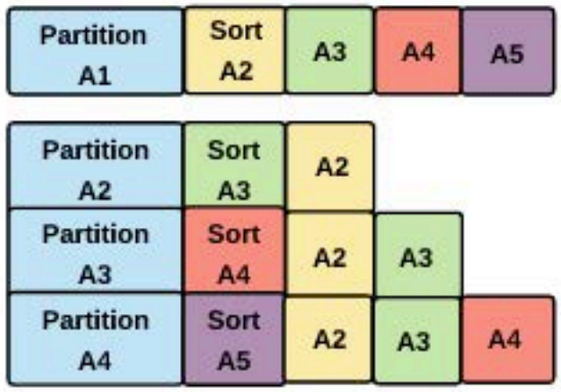
- Introduction
- Auto Scaling Deployment Concepts
- Deployment Concepts With EC2
- CloudWatch For DevOps
- CloudFormation For DevOps
- Elastic Beanstalk For DevOps
- Application Deployments With OpsWorks
- DynamoDB Concepts
- S3 Concepts For DevOps
- Blue/Green Deployments
- Scenario Solver
- Deployment Pipelines
- API Gateway
- Lambda

X

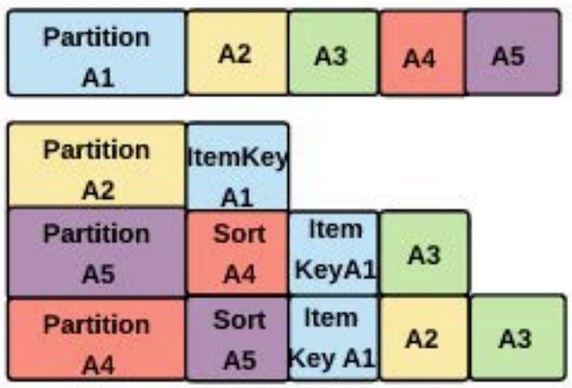
AWS

Local And Global Secondary Indexes

Local Secondary Indexes



Global Secondary Indexes



Secondary Indexes

- A secondary index is a data structure that contains a subset of attributes from a table, along with an alternate key to support Query operations. A table can have multiple secondary indexes, which gives your applications access to many different query patterns.
- Global Secondary Index - an index with a partition key and a sort key that can be different from those on the base table.
- Local Secondary Index - an index that has the same partition key as the base table, but a different sort key.

Choose a Topic

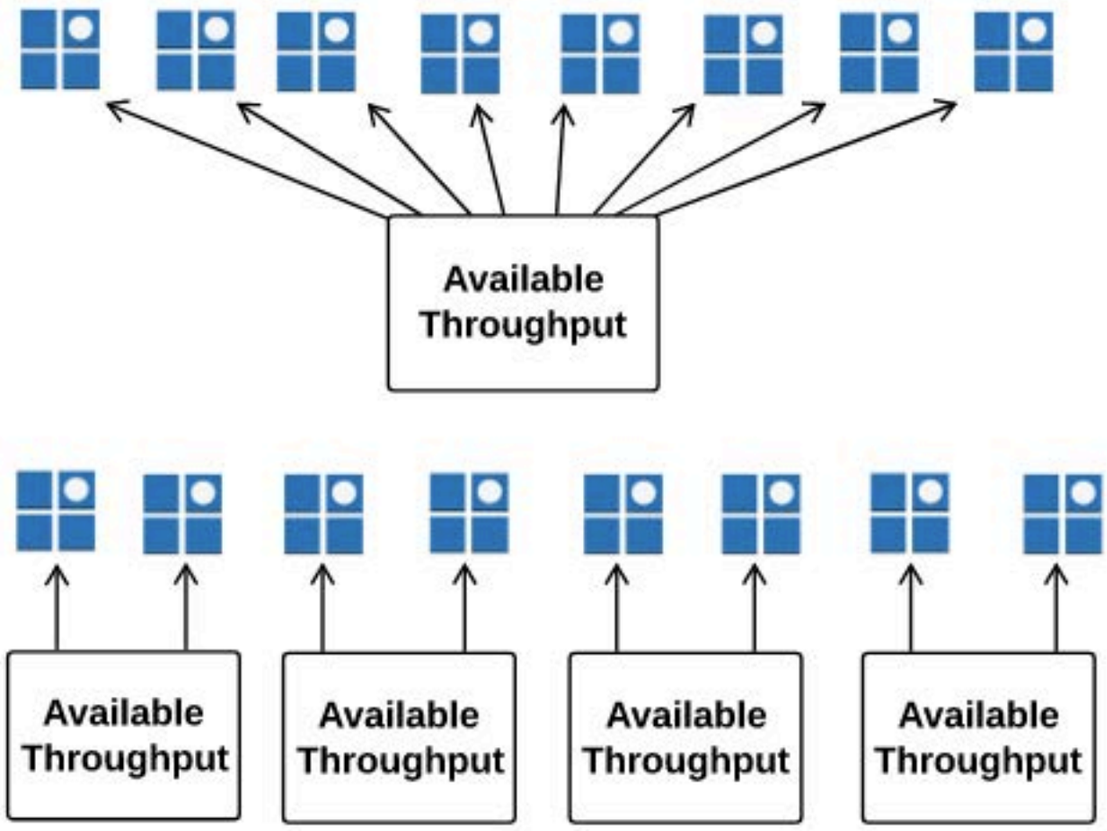
- Introduction
- Auto Scaling Deployment Concepts
- Deployment Concepts With EC2
- CloudWatch For DevOps
- CloudFormation For DevOps
- Elastic Beanstalk For DevOps
- Application Deployments With OpsWorks
- DynamoDB Concepts
- S3 Concepts For DevOps
- Blue/Green Deployments
- Scenario Solver
- Deployment Pipelines
- API Gateway
- Lambda

Key Concepts

X

AWS

Understanding Provisioned Throughput



Provisioned Throughput

- Provisioned Capacity
  - Unit of read capacity: 1 strongly consistent read/second or 2 eventually consistent reads/second for items as large as 4 KB.
  - Unit of write capacity: 1 write/second for items up to 1KB.
  - Key concepts:
    - Calculating required throughput
    - Understanding how secondary indexes affect throughput
    - What happens if your apps read/writes exceed throughput?
- Calculating Read Capacity - Round up to the nearest 4KB multiplier. Items that are 3KB in size can still only do 1 strongly consistent or 2 eventually consistent reads per second.

Choose a Topic

- Introduction
- Auto Scaling Deployment Concepts
- Deployment Concepts With EC2
- CloudWatch For DevOps
- CloudFormation For DevOps
- Elastic Beanstalk For DevOps
- Application Deployments With OpsWorks
- DynamoDB Concepts
- S3 Concepts For DevOps
- Blue/Green Deployments
- Scenario Solver
- Deployment Pipelines
- API Gateway
- Lambda

X

AWS

Provisioned Throughput - continued

- Read Capacity Example: Your items are 3 KB and you want to read 80 (strongly consistent read) items from a table per second:

80 * (3KB (round up to 4)/4 KB = 80 * 1 = 80 provisioned throughput required

For eventually consistent: (80 * 1)/2 = 40 required read capacity

- Calculating Write Capactiy
  - Round up to the nearest 1KB multiplier
  - Formula: (ITEM SIZE(rounded up to the nearest 1KB * # of items)

Example: Item size 1.5 KB, want to write 10 items per second

10 * (1.5 KB (round up to 2) / 1 KB = 10 * 2 = 20 provisioned write throughput

Read Throughput with Local Secondary Indexes

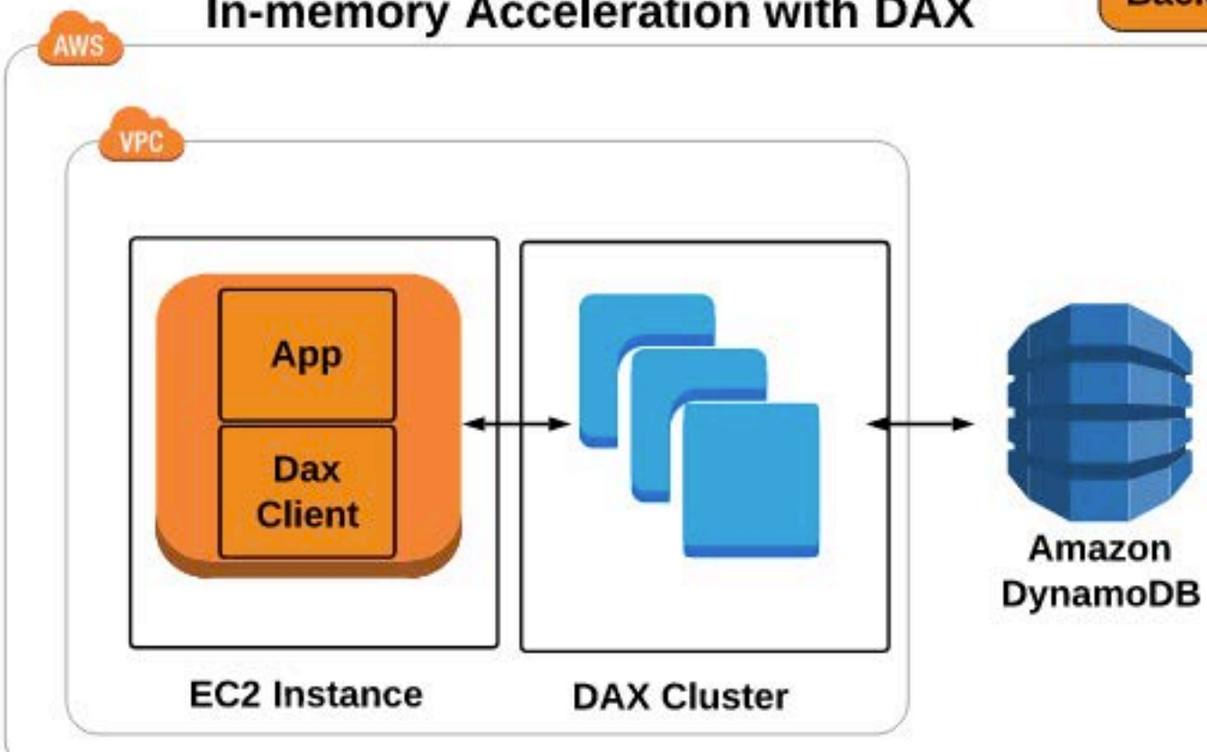
- Uses the same read/write capacity from parent table.
- If you read only index keys and projected attributes, the calculations are the same.
  - You calculate using the size of the index entry, not the table item size.
  - Rounded up to the nearest 4 KB
- Write Throughput With Local Secondary Indexes
  - Adding, Updating, or Deleting an item in a table also costs write capacity units to perform the action on the local index.
- Read Throughput With Global Secondary Indexes
  - Global indexes have their own throughput capacity, completely separate from that of the table's capacity.
  - Support eventually consistent reads, which means that a single global secondary index query can get up to 8KB read capacity unit.
- Write Throughput With Global Secondary Indexes
  - Putting, Updating, or Deleting items in a table consumes the index write capacity units
- Exceeding Throughput
  - Requests exceeding the allocated throughput may be throttled
  - With global secondary indexes, all indexes must have enough write capacity or the write might get throttled.
  - You can monitor throughput in the AWS Console.

Choose a Topic

- Introduction
- Auto Scaling Deployment Concepts
- Deployment Concepts With EC2
- CloudWatch For DevOps
- CloudFormation For DevOps
- Elastic Beanstalk For DevOps
- Application Deployments With OpsWorks
- DynamoDB Concepts
- S3 Concepts For DevOps
- Blue/Green Deployments
- Scenario Solver
- Deployment Pipelines
- API Gateway
- Lambda

Back

In-memory Acceleration with DAX



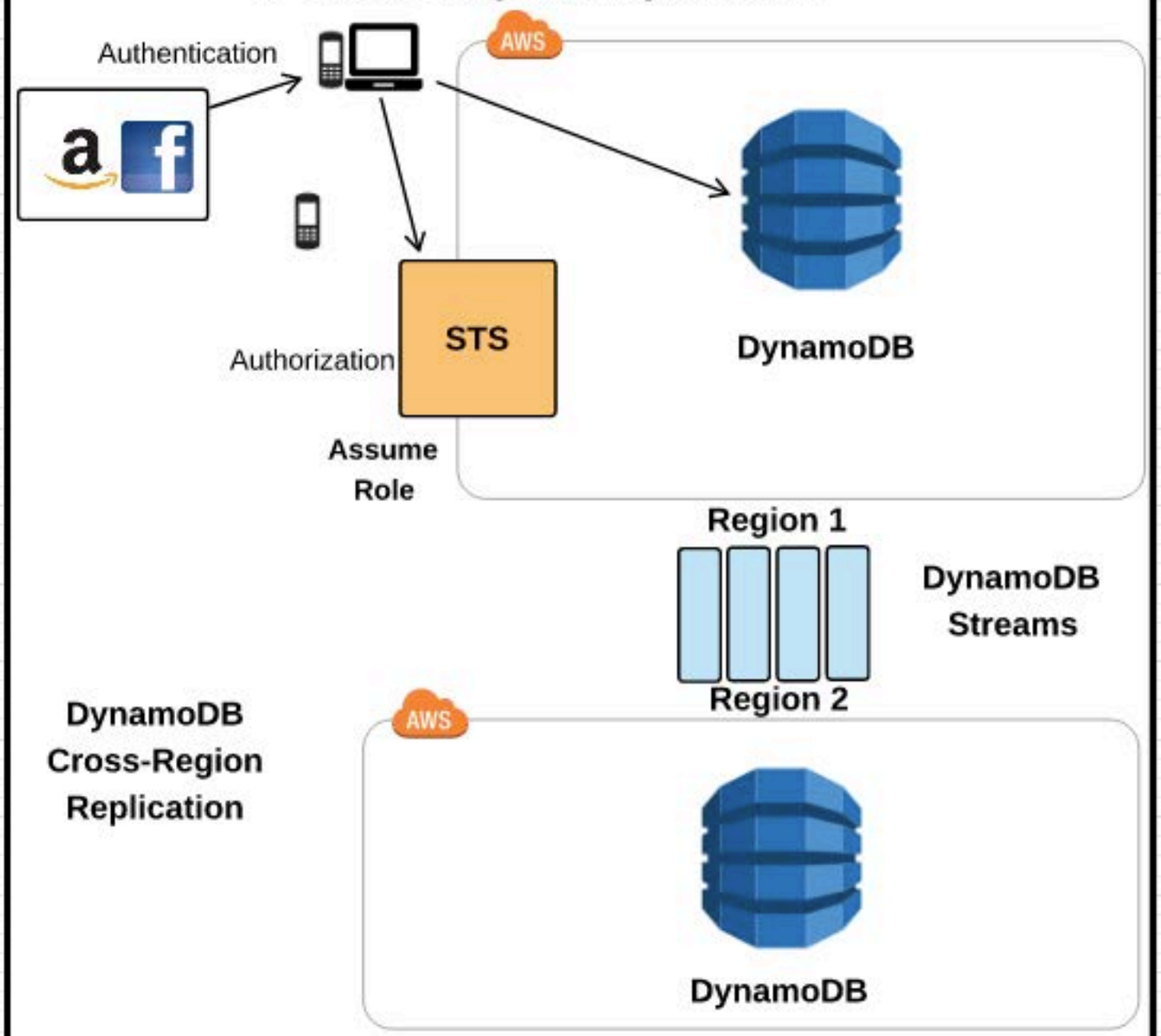
In-Memory Acceleration with DAX

- Amazon DynamoDB is designed for scale and performance.
- Response times can be measured in single digit milliseconds.
- However, there are certain use cases that require response times in microseconds.
- DAX delivers fast response times for accessing eventually consistent data. DAX enables you to benefit from fast in-memory performance for demanding applications.
- DAX Use Cases:
  - Apps that require the fastest possible response time for reads.
  - Apps that read a small number of items more frequently than others.
  - Apps that are read-intensive, but are also cost-sensitive.
  - Applications that require repeated reads against a large set of data.

Choose a Topic

- Introduction
- Auto Scaling Deployment Concepts
- Deployment Concepts With EC2
- CloudWatch For DevOps
- CloudFormation For DevOps
- Elastic Beanstalk For DevOps
- Application Deployments With OpsWorks
- DynamoDB Concepts
- S3 Concepts For DevOps
- Blue/Green Deployments
- Scenario Solver
- Deployment Pipelines
- API Gateway
- Lambda

Web Identity Federation & Cross-Region Replication



DynamoDB Cross-Region Replication

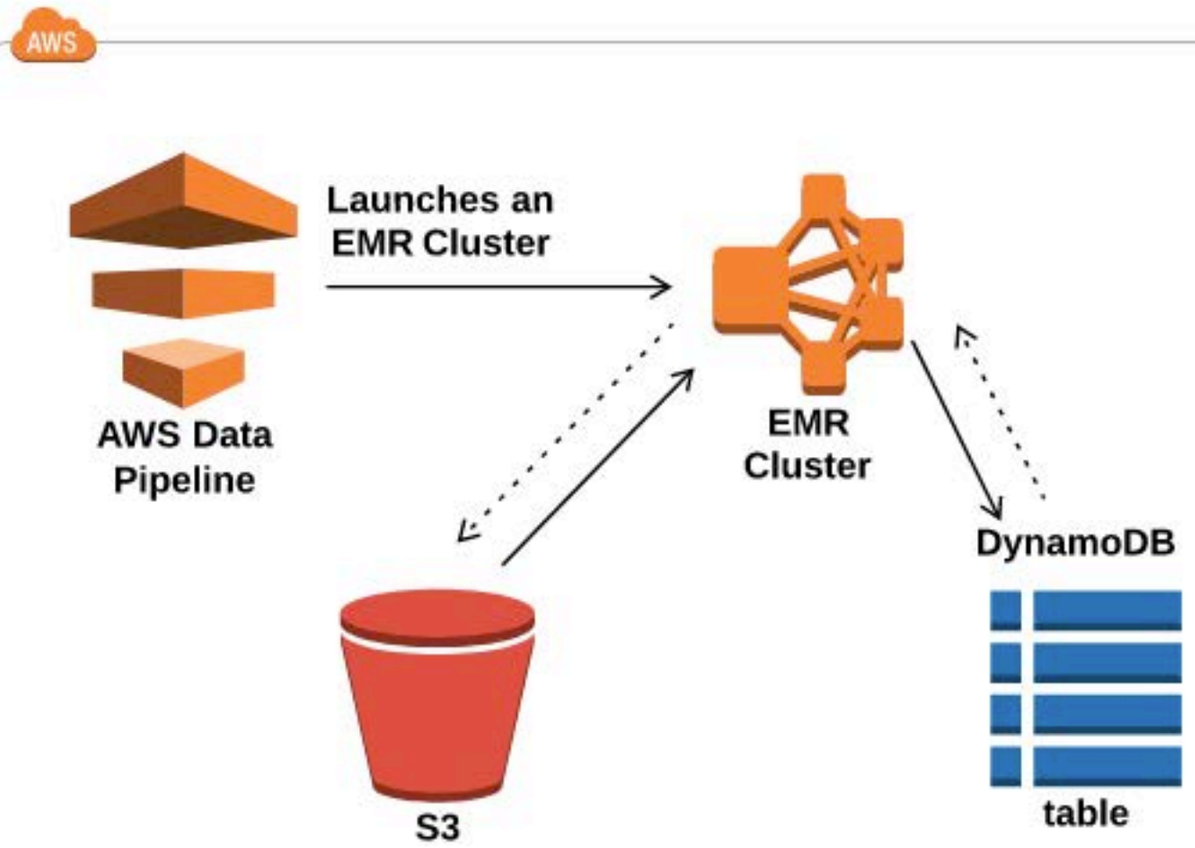
- Allows identical copies (called replicas) of a DynamoDB table (called master table) to be maintained in one or more AWS regions
- writes to the table will be automatically propagated to all replicas
- Read Replicas are updated asynchronously.
- Use Cases: DR, Faster Reads, Data Migration
- Used Data Pipeline before DynamoDB Streams was released.

Choose a Topic

- Introduction
- Auto Scaling Deployment Concepts
- Deployment Concepts With EC2
- CloudWatch For DevOps
- CloudFormation For DevOps
- Elastic Beanstalk For DevOps
- Application Deployments With OpsWorks
- DynamoDB Concepts
- S3 Concepts For DevOps
- Blue/Green Deployments
- Scenario Solver
- Deployment Pipelines
- API Gateway
- Lambda

Data Pipeline With DynamoDB

X



Using Data Pipeline For Import/Export With DynamoDB

- You can use Data Pipeline to export data from a DynamoDB table to a file in S3.
- You can import data from S3 into a DynamoDB table, in the same AWS region or in a different region.
- Copy data from a DynamoDB table in one region, store the data in Amazon S3, and import the data from S3 to an identical DynamoDB table in a second region.
- When you export or import DynamoDB data, you will incur additional costs for the underlying AWS services that are used.
- When you use Data Pipeline for exporting and importing data, you must specify the actions the pipeline can perform, and which resources the pipeline can consume.
- Required IAM Roles:
  - DataPipelineDefaultRole - actions your pipeline can take on your behalf.
  - DataPipelineDefaultResourceRole - for the resources pipeline will provision on your behalf (EMR Cluster and the EC2 instances within it).

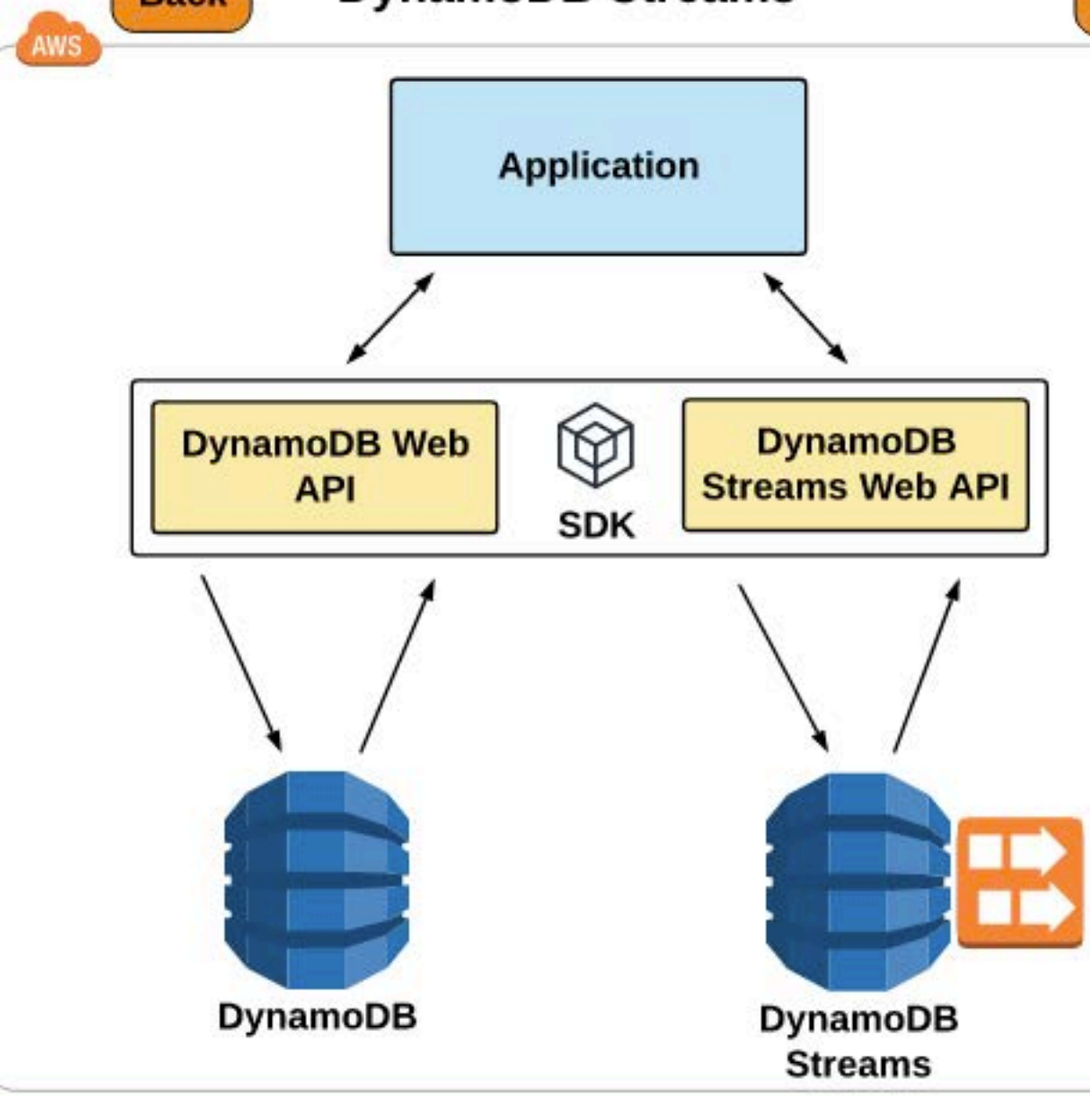
Choose a Topic

- Introduction
- Auto Scaling Deployment Concepts
- Deployment Concepts With EC2
- CloudWatch For DevOps
- CloudFormation For DevOps
- Elastic Beanstalk For DevOps
- Application Deployments With OpsWorks
- DynamoDB Concepts
- S3 Concepts For DevOps
- Blue/Green Deployments
- Scenario Solver
- Deployment Pipelines
- API Gateway
- Lambda

Back

DynamoDB Streams

Next



DynamoDB Streams

- DynamoDB Streams captures a time-ordered sequence of item-level modifications in any DynamoDB table, and stores this information in a log for up to 24 hours.
- Applications can access this log and view the data items as they appeared before and after they were modified, in near real time.
- When you enable a stream on a table, DynamoDB captures information about every modification to data items in the table.
- The near real-time aspect enables building applications that consume these streams and take action based on the contents.
- AWS maintains separate endpoints for DynamoDB and DynamoDB Streams.

Choose a Topic

- Introduction
- Auto Scaling Deployment Concepts
- Deployment Concepts With EC2
- CloudWatch For DevOps
- CloudFormation For DevOps
- Elastic Beanstalk For DevOps
- Application Deployments With OpsWorks
- DynamoDB Concepts
- S3 Concepts For DevOps
- Blue/Green Deployments
- Scenario Solver
- Deployment Pipelines
- API Gateway
- Lambda

Back

DynamoDB Streams - Continued

AWS

- To work with database tables and indexes, your application will need to access a DynamoDB endpoint. To read and process DynamoDB Streams records, your application will need to access a DynamoDB Streams endpoint in the same region.
- You can enable a stream on a new table when you create it. You can also enable or disable a stream on an existing table, or change the settings of a stream.
- DynamoDB Streams operates asynchronously, so there is no performance impact on a table if you enable a stream.
- To read and process a stream, your application will need to connect to a DynamoDB Streams endpoint and issue API requests.
- A stream consists of *stream records*. Each stream record represents a single data modification in the DynamoDB table to which the stream belongs.
- All data in DynamoDB Streams is subject to a 24 hour lifetime. You can retrieve and analyze the last 24 hours of activity for any given table; however, data older than 24 hours is susceptible to trimming (removal) at any moment.