

DAT321

Amazon DynamoDB Under the Hood: How We Built a Hyper-Scale Database

Jaso Sorenson
Senior Principal Engineer
AWS/DynamoDB

aws
re:Invent

© 2018, Amazon Web Services, Inc. or its affiliates. All rights reserved.



Come to this session to learn how **Amazon DynamoDB was built as the hyper-scale database for internet-scale applications**. In January 2012, Amazon launched DynamoDB, a cloud-based No-SQL database service designed from the ground up to support extreme scale, with the security, availability, performance, and manageability needed to run mission-critical workloads. This session discloses for the first time the underpinnings of DynamoDB, and how we run a fully managed non-relational database used by more than 100,000 customers. We cover the underlying technical aspects of how an application works with DynamoDB for authentication, metadata, storage nodes, streams, backup, and global replication.

Onboarding to Amazon DynamoDB

What is the Goal?

Learn about features

Understand tools

Use DynamoDB more effectively

Agenda

GetItem / PutItem

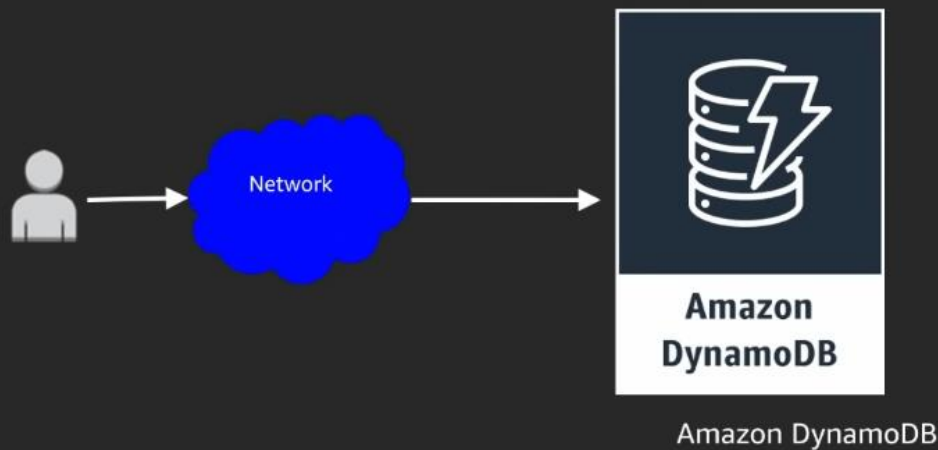
Auto Scaling

Backup Restore

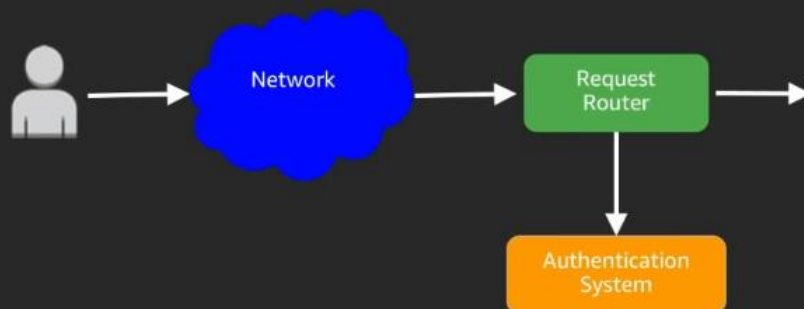
Streams

Global Tables

GetItem



GetItem (Step 1)



The Request Router is the public facing API for DynamoDB that does things like authentication of the request's caller to make sure they can do what they want to do. We use this same Authentication system for all our AWS services at the entry point.

Sample Policy

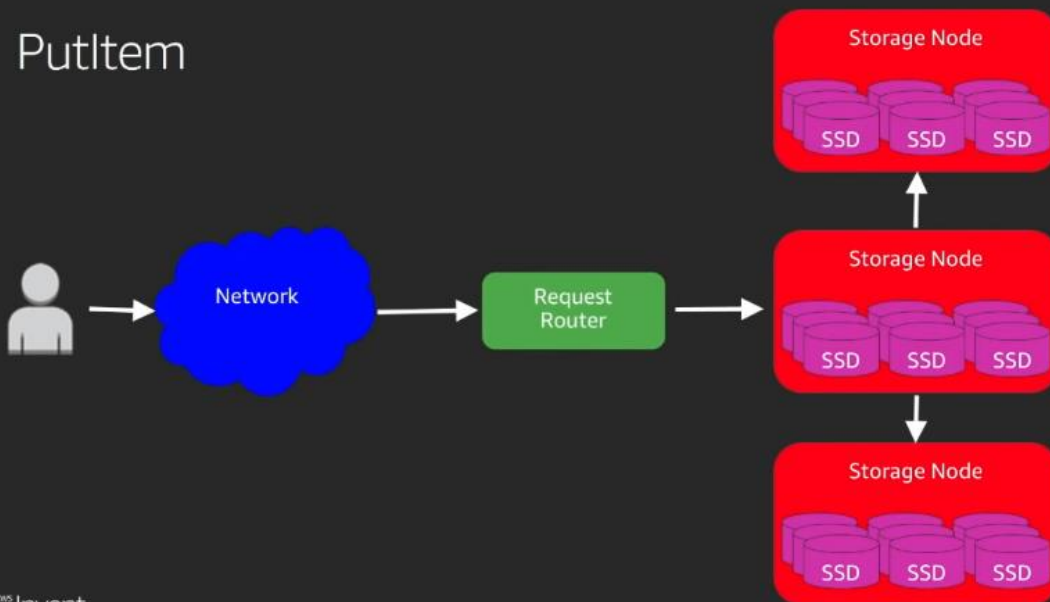
```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "dynamodb:GetItem"
      ],
      "Resource": [
        "arn:aws:dynamodb:us-west-2:1234567890:table/customer"
      ],
      "Condition": {
        "ForAllValues:StringEquals": {
          "dynamodb:LeadingKeys": [
            "${www.amazon.com:user_id}"
          ]
        }
      }
    }
  ]
}
```

GetItem (Step 2)



It then makes a call out to our storage nodes for the actual data you are asking for.

PutItem



A Put is a little bit more complicated because we need to durably store the data by writing to 3 storage nodes and waits for 1 node to acknowledge the request was successfully stored.

DynamoDB evolved from Dynamo

Dynamo: Amazon's Highly Available Key-value Store

Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall and Werner Vogels

Amazon.com

DynamoDB has evolved from the Dynamo described in this paper, but instead used Paxos

Paxos

The Part-Time Parliament

Leslie Lamport

September 1, 1989

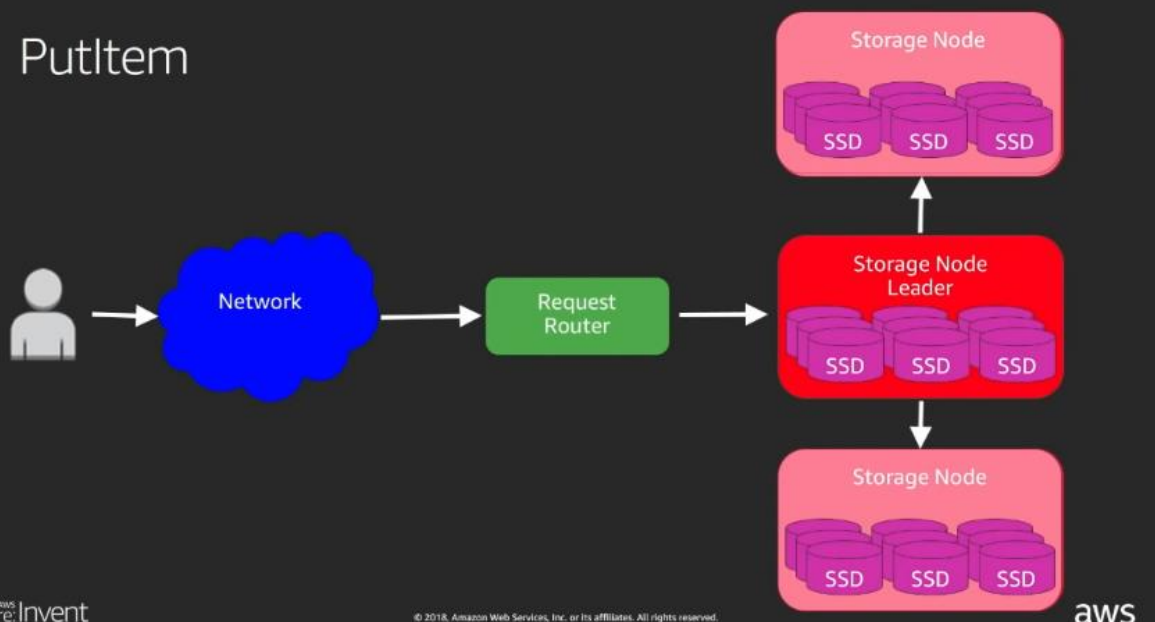
Paxos Made Simple

Leslie Lamport

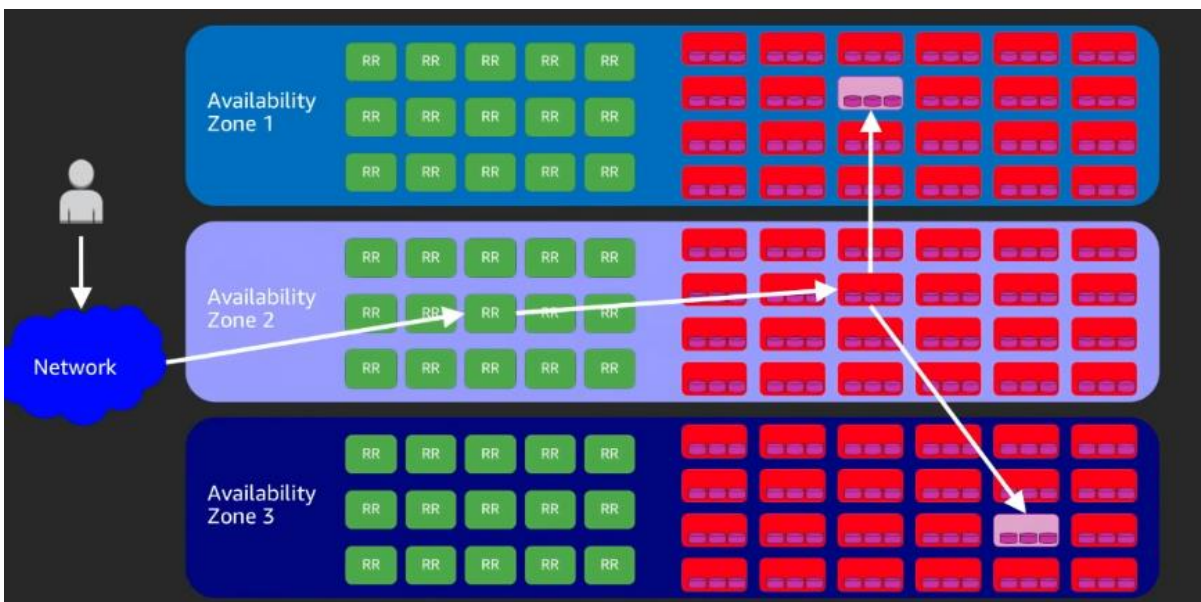
01 Nov 2001

Paxos is an algorithm for to make a set of nodes agree on something

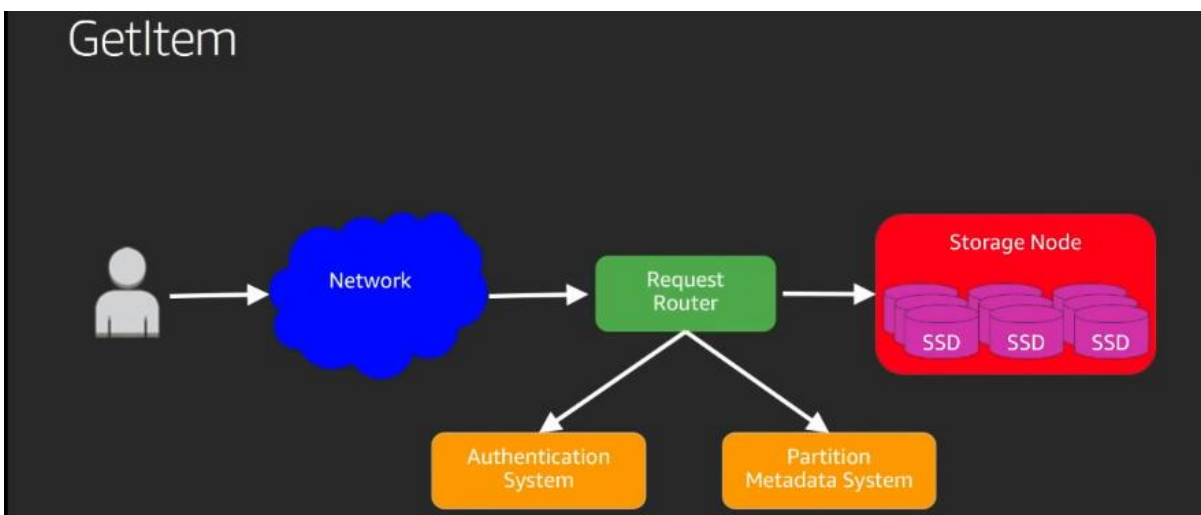
PutItem



All the DynamoDB storage nodes run Paxos and we are trying to get agreement on a leader for a partition or table. The Leader is always up-to-date and accepts all the data writes and propagates the data to other follower nodes for the partition. A leader is always sending back a periodic heartbeat, failure to do so will cause a new leader election to take place.



A Request Router (RR) is stateless so it doesn't matter which one of the RR boxes the request gets routed through. The RR then makes a call to the leader storage node of the partition the request needs to talk to and the leader gets to talk to the other related follower nodes for data replication.



The Partition Metadata Service is a service that holds the leader-follower nodes information for all the data partitions

Tables

CustID	Customer Information
145783	{ name:"Bob", city:"London", ...}
236294	{ name:"Sara", city:"Tampa", ...}
333363	{ name:"Betty", city:"Madison", ...}
445104	{ name:"James", city:"Miami", ...}
523422	{ name:"Alex", city:"London", ...}
643145	{ name:"Val", city:"Seattle", ...}
723342	{ name:"Jeff", city:"Toledo", ...}

When you make a table in DynamoDB, you need to tell it your primary hash key for the table like the CustID above.

Hashing

Hash Value	CustID	Customer Information
0x9531	145783	{ name:"Bob", city:"London", ...}
0x12A8	236294	{ name:"Sara", city:"Tampa", ...}
0x6134	333363	{ name:"Betty", city:"Madison", ...}
0x3391	445104	{ name:"James", city:"Miami", ...}
0xF355	523422	{ name:"Alex", city:"London", ...}
0xB082	643145	{ name:"Val", city:"Seattle", ...}
0xEA8A	723342	{ name:"Jeff", city:"Toledo", ...}

DynamoDB then computes a hash for the primary key

Partitioning

Hash Value	CustID	Customer Information
0x9531	145783	{ name:"Bob", city:"London", ...}
0x12A8	236294	{ name:"Sara", city:"Tampa", ...}
0x6134	333363	{ name:"Betty", city:"Madison", ...}
0x3391	445104	{ name:"James", city:"Miami", ...}
0xF355	523422	{ name:"Alex", city:"London", ...}
0xB082	643145	{ name:"Val", city:"Seattle", ...}
0xEA8A	723342	{ name:"Jeff", city:"Toledo", ...}

0x12A8	236294	{ name:"Sara", city:"Tampa", ...}
0x3391	445104	{ name:"James", city:"Miami", ...}
0x6134	333363	{ name:"Betty", city:"Madison", ...}

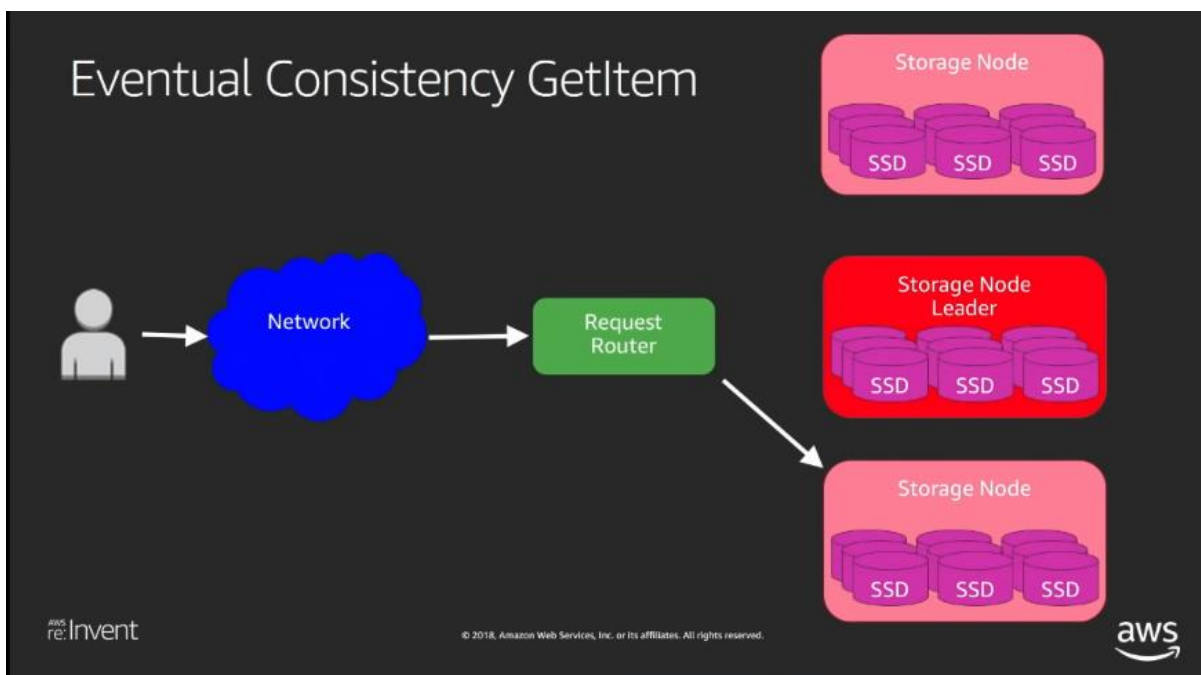
0x9531	145783	{ name:"Bob", city:"London", ...}
0xB082	643145	{ name:"Val", city:"Seattle", ...}

0xEA8A	723342	{ name:"Jeff", city:"Toledo", ...}
0xF355	523422	{ name:"Alex", city:"London", ...}

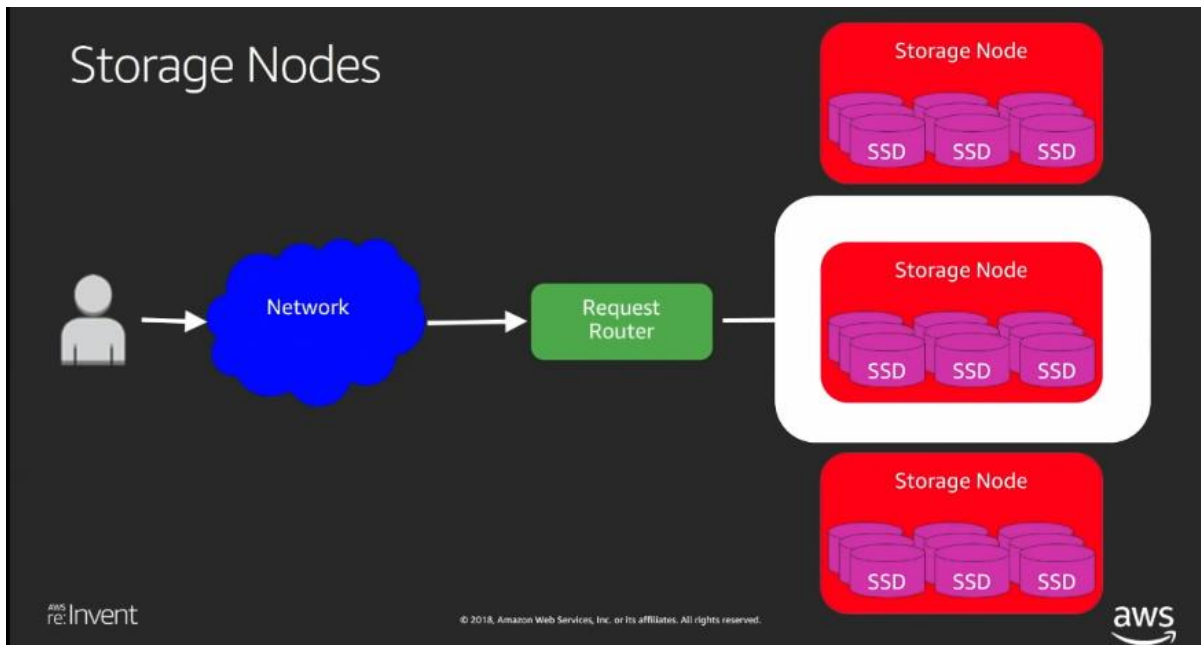
Then uses the hash to sort the data into partitions as above



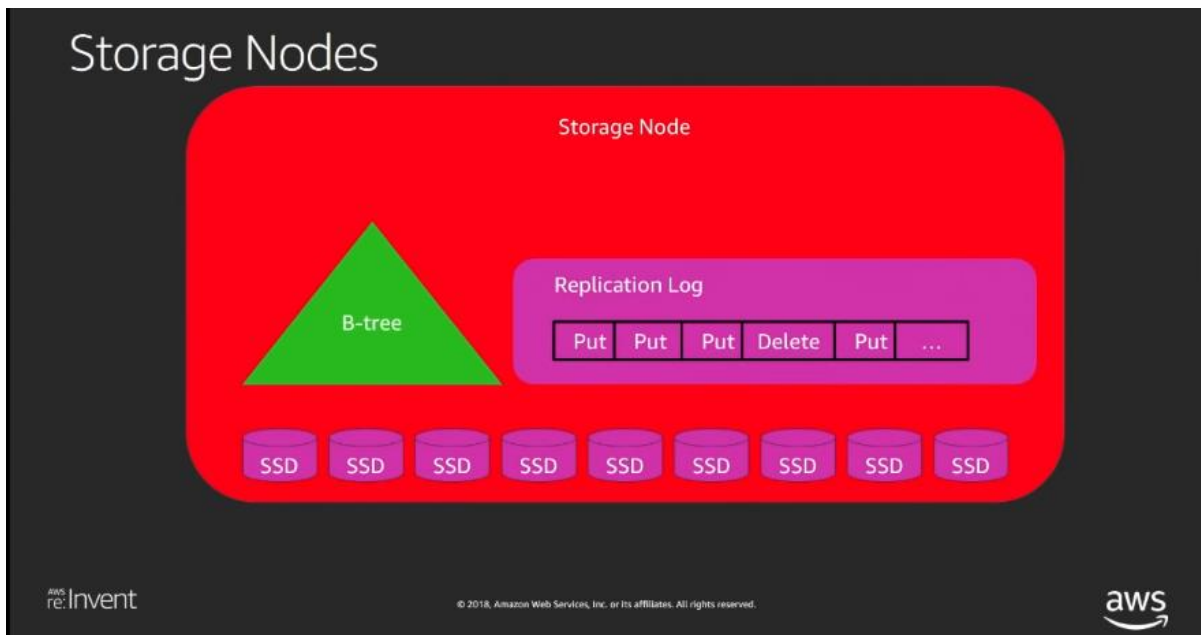
The partitions then get mapped out to the storage nodes for the AZ the customer specifies.



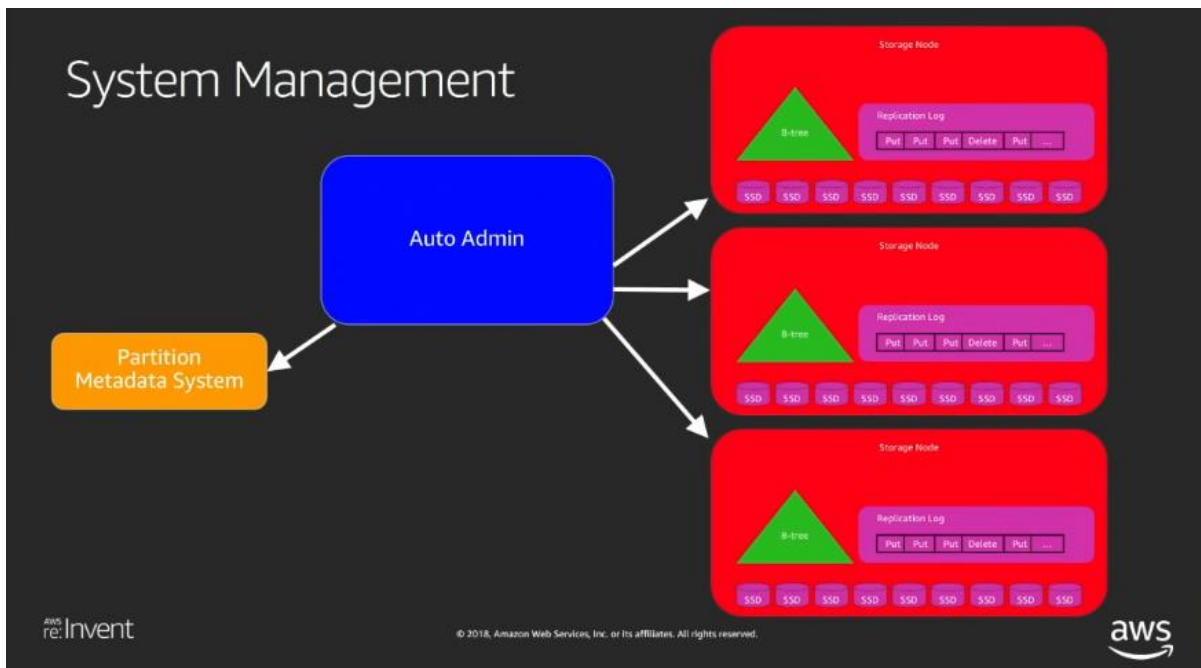
For eventually consistent read requests, we let the RR randomly choose the storage nodes holding the data, this means that you may not get the most recent version of your data in some cases.



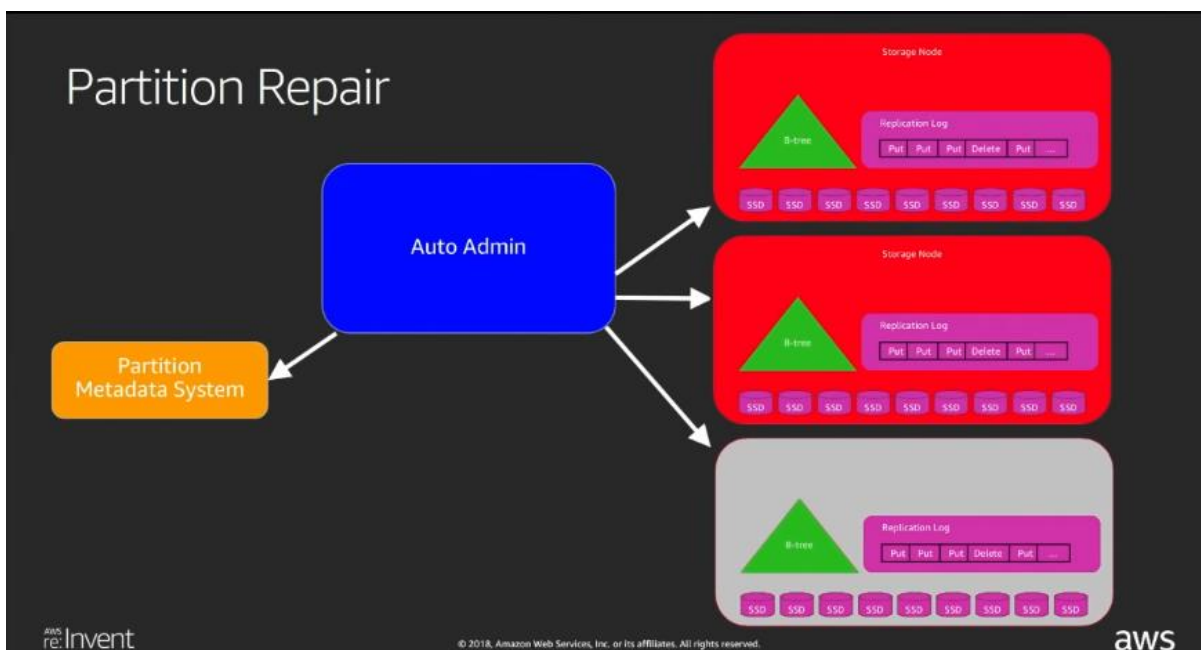
What happens inside the storage nodes?



The B-Tree is where we do all the querying and scanning for the user data using the resulting B-Tree index. the replication log is recording every mutation to the data partition.

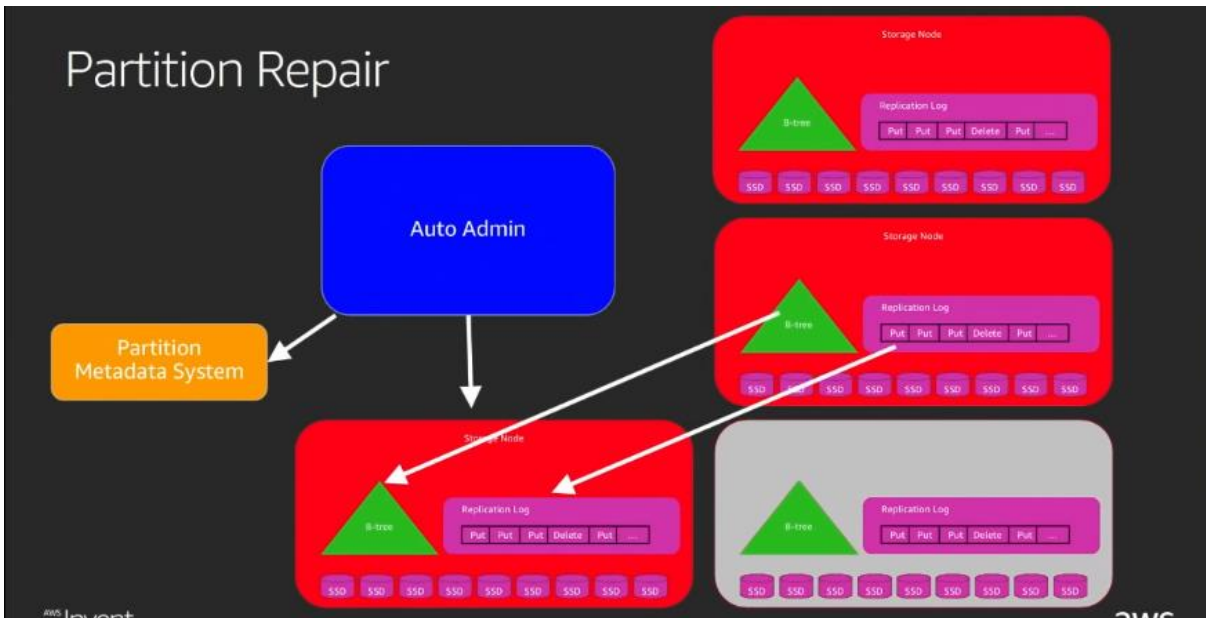


DynamoDB has a component called Auto Admin that does many things, like making sure the PMS is up-to-date with the leader location for all partitions.



Auto-Admin also does partition repair using another storage node to take over that failed partition as below

Partition Repair



It then starts copying the B-Tree and replication log over to the new storage node, then it makes sure the replication log is applied to the new B-Tree to make sure the new node catches up to the leader.

Secondary Index

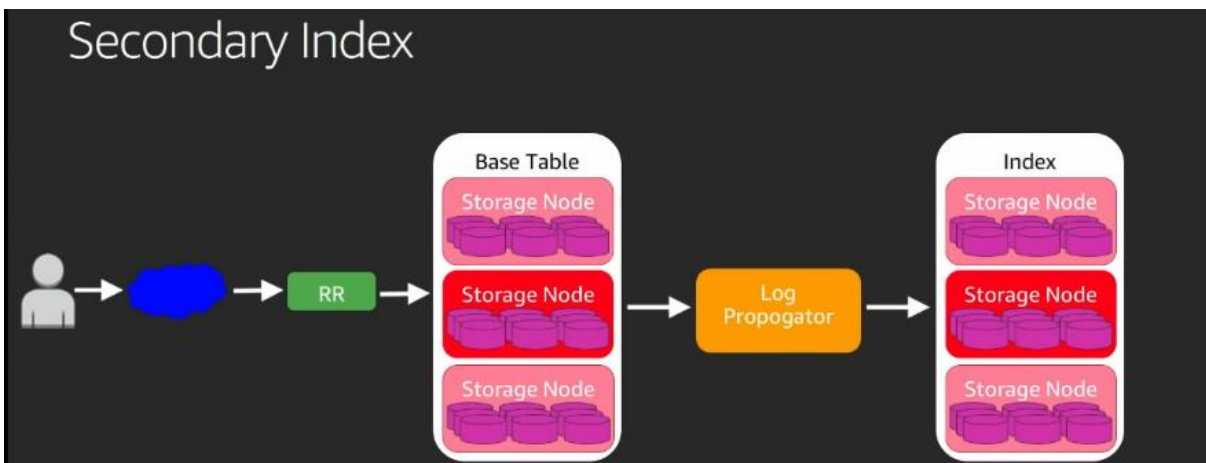
Hash Value	Name	Projected Attributes
0x7232	Bob	{ name:"Bob", ... }
0x45B3	Sara	{ name:"Sara", ... }
0x99A5	Betty	{ name:"Betty", ... }
0x14C7	James	{ name:"James", ... }
0xA3B5	Alex	{ name:"Alex", ... }
0x04CA	Val	{ name:"Val", ... }
0xDA8A	Jeff	{ name:"Jeff", ... }

0x04CA	Val	{ name:"Val", ... }
0x14C7	James	{ name:"James", ... }
0x45B3	Sara	{ name:"Sara", ... }
0x7232	Bob	{ name:"Bob", ... }

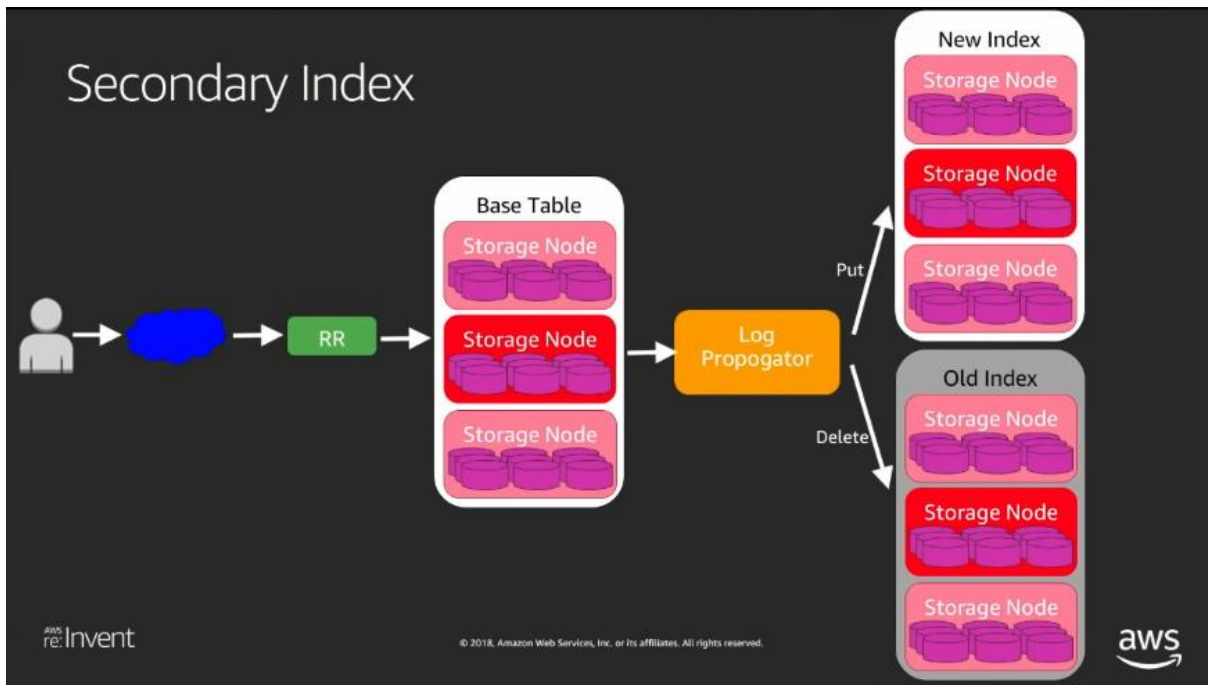
0x99A5	Betty	{ name:"Betty", ... }
0xA3B5	Alex	{ name:"Alex", ... }
0xDA8A	Jeff	{ name:"Jeff", ... }

We can also build a secondary index on the name attribute for our item being stored in the table. The SI are independent of what happens with the base table.

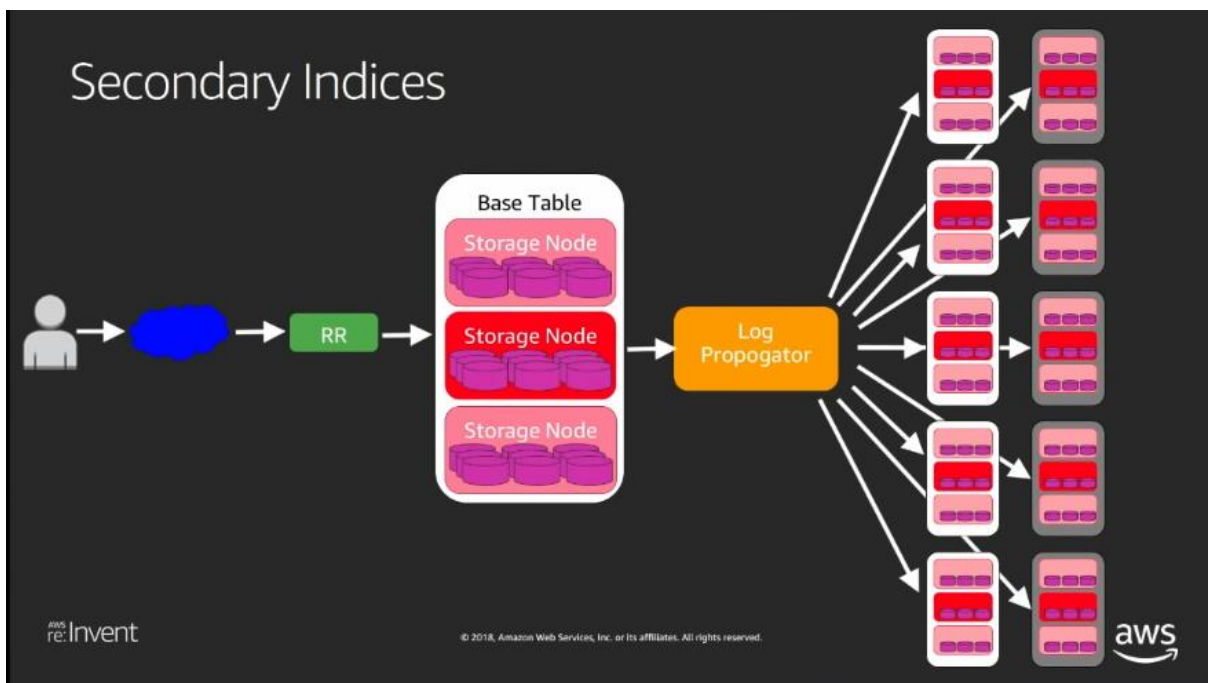
Secondary Index



The Log Propagator is watching the repl log and the storage nodes and executing a PUT when a secondary index changes on an SI partition



We update the value and rewrite the new value somewhere else



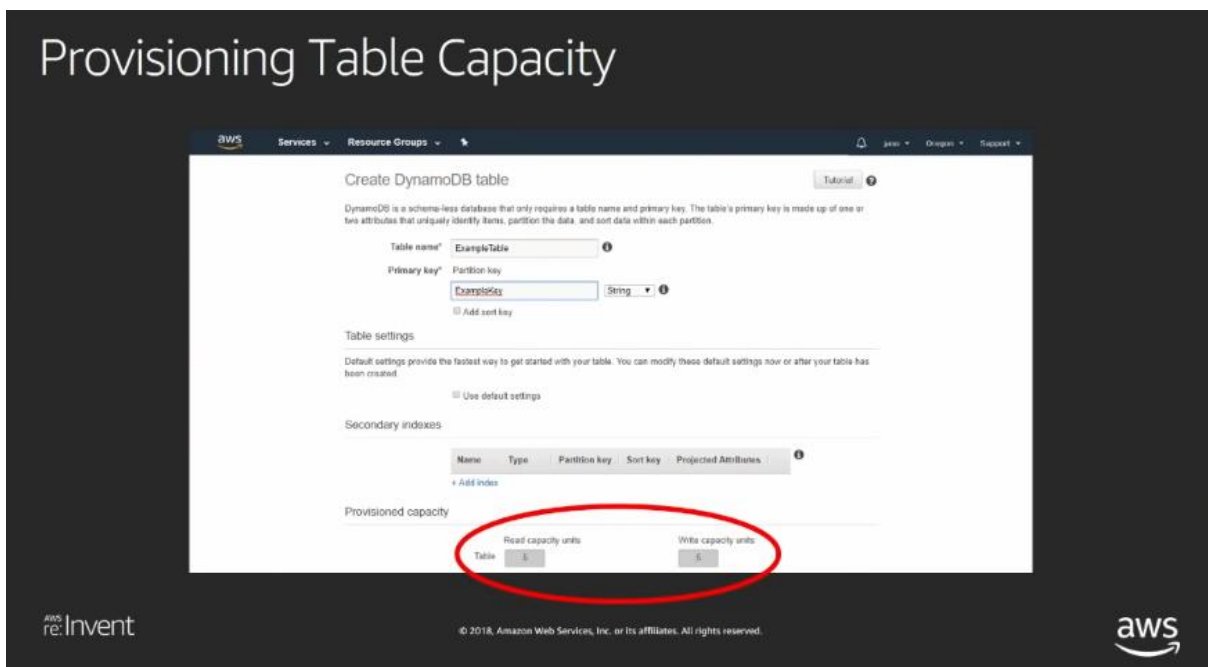
You can have up to 5 secondary indexes, this creates many possible updates and changes for changes



This is like the DBA for DynamoDB that does a lot of things.



This is about provisioning new tables in DynamoDB.



You need to provide the table name, primary key, and the read and write capacity units. 1 read capacity unit, RCU will allow you to read up to a 4k object.

Provisioning Trade-off

Functionality



Provisioning Example

300 Read Capacity Units (RCU)

Ignoring Writes

This is identical for the write side

Provisioning

0x12A8	236294	{ name:"Sara", city:"Tampa", ...}
0x3391	445104	{ name:"James", city:"Miami", ...}
0x6134	333363	{ name:"Betty", city:"Madison", ...}

← 100 RCUs

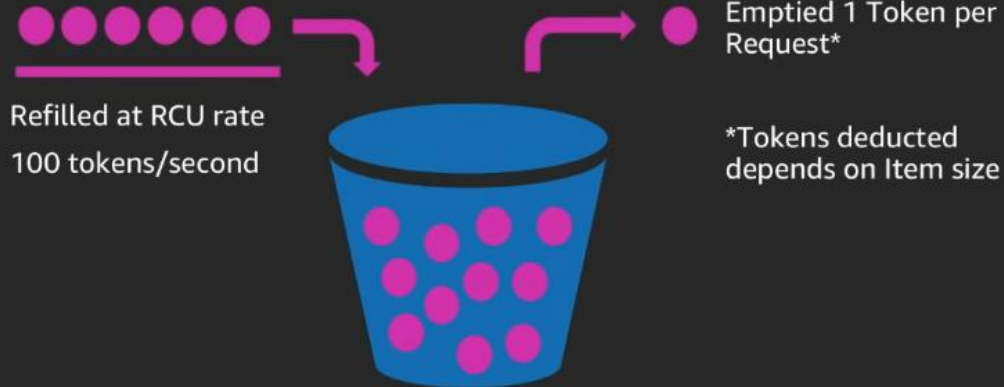
0x9531	145783	{ name:"Bob", city:"London", ...}
0xB082	643145	{ name:"Val", city:"Seattle", ...}

← 100 RCUs

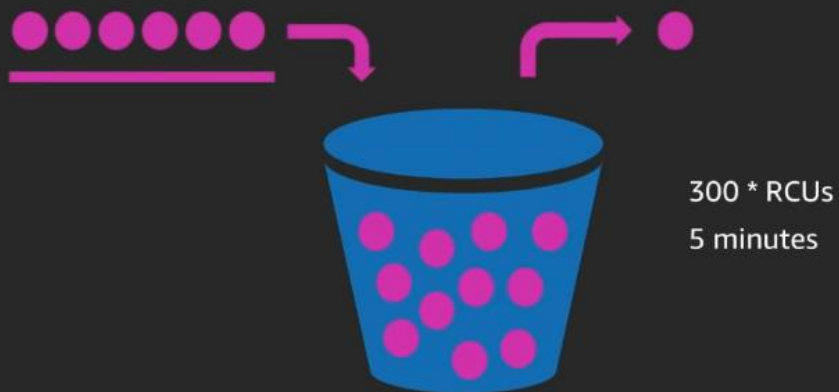
0xEA8A	723342	{ name:"Jeff", city:"Toledo", ...}
0xF355	523422	{ name:"Alex", city:"London", ...}

← 100 RCUs

Token Bucket Algorithm



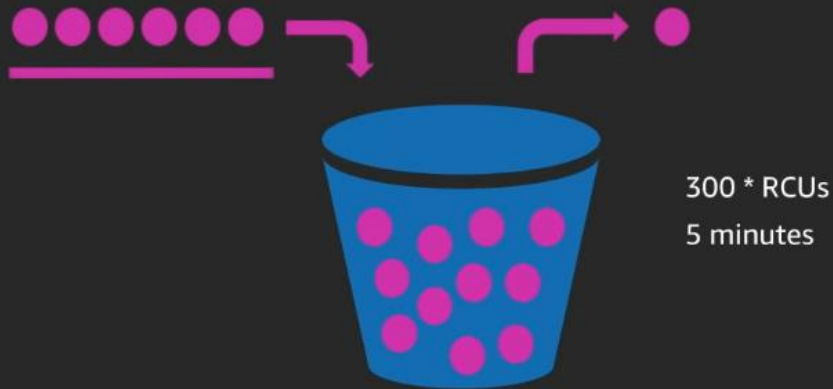
Token Bucket Capacity



Full Token Bucket



Partition Bursting



Unbalanced Load

0x12A8	236294	{ name:"Sara", city:"Tampa", ...}
0x3391	445104	{ name:"James", city:"Miami", ...}
0x6134	333363	{ name:"Betty", city:"Madison", ...}

25 RCU attempted

0x9531	145783	{ name:"Bob", city:"London", ...}
0xB082	643145	{ name:"Val", city:"Seattle", ...}

150 RCU attempted

0xEA8A	723342	{ name:"Jeff", city:"Toledo", ...}
0xF355	523422	{ name:"Alex", city:"London", ...}

50 RCU attempted

Unbalanced Load

75 RCU unused

0x12A8	236294	{ name:"Sara", city:"Tampa", ...}
0x3391	445104	{ name:"James", city:"Miami", ...}
0x6134	333363	{ name:"Betty", city:"Madison", ...}

25 RCU admitted

OK

0x9531	145783	{ name:"Bob", city:"London", ...}
0xB082	643145	{ name:"Val", city:"Seattle", ...}

100 RCU admitted

50 throttles
per second

0xEA8A	723342	{ name:"Jeff", city:"Toledo", ...}
0xF355	523422	{ name:"Alex", city:"London", ...}

50 RCU admitted

OK

Hot partitions will get throttled as above

Adaptive Capacity



Refilled at RCU rate
100 tokens/second
150 tokens/second



Adaptive Capacity



Refilled at RCU rate
150 tokens/second



Adaptive Capacity
Multiplier = 1.5

Adaptive Capacity Active

75 RCU unused

0x12A8	236294	{ name:"Sara", city:"Tampa", ...}
0x3391	445104	{ name:"James", city:"Miami", ...}
0x6134	333363	{ name:"Betty", city:"Madison", ...}

25 RCU admitted OK

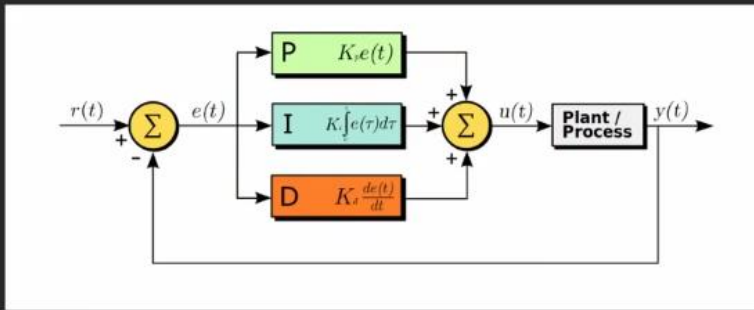
0x9531	145783	{ name:"Bob", city:"London", ...}
0xB082	643145	{ name:"Val", city:"Seattle", ...}

150 RCU admitted OK

0xEA8A	723342	{ name:"Jeff", city:"Toledo", ...}
0xF355	523422	{ name:"Alex", city:"London", ...}

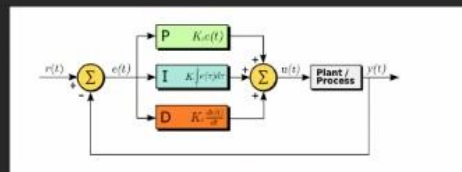
50 RCU admitted OK

PID Controller



<https://commons.wikimedia.org/wiki/File:PID.svg>

PID Controller



<https://commons.wikimedia.org/wiki/File:PID.svg>

PID inputs:

PID output:

Consumed Capacity
Provisioned Capacity
Throttling Rate
Current Multiplier

New Multiplier

We apply this to the partitions in your table to determine the adaptive capacity to apply at each time

Too Much Allowed

150 RCU over provisioned

0x12A8	236294	{ name:"Sara", city:"Tampa", ...}
0x3391	445104	{ name:"James", city:"Miami", ...}
0x6134	333363	{ name:"Betty", city:"Madison", ...}

150 RCU admitted

0x9531	145783	{ name:"Bob", city:"London", ...}
0xB082	643145	{ name:"Val", city:"Seattle", ...}

150 RCU admitted

0xEA8A	723342	{ name:"Jeff", city:"Toledo", ...}
0xF355	523422	{ name:"Alex", city:"London", ...}

150 RCU admitted

Partition Bursting



Refilled at RCU rate
100 tokens/second



Adaptive Capacity

~~Multiplier = 1.5~~

Multiplier = 1.0

Throttling

0x12A8	236294	{ name:"Sara", city:"Tampa", ...}
0x3391	445104	{ name:"James", city:"Miami", ...}
0x6134	333363	{ name:"Betty", city:"Madison", ...}

100 RCU admitted **50 Throttled**

0x9531	145783	{ name:"Bob", city:"London", ...}
0xB082	643145	{ name:"Val", city:"Seattle", ...}

100 RCU admitted **50 Throttled**

0xEA8A	723342	{ name:"Jeff", city:"Toledo", ...}
0xF355	523422	{ name:"Alex", city:"London", ...}

100 RCU admitted **50 Throttled**

Auto Scaling

customer Close

Overview Items Metrics Alarms **Capacity** Indexes Global Tables Backups Triggers Access control Tags

Scaling activities

Provisioned capacity

Read capacity units

Table 300

name-index 300

Write capacity units

5

5

Estimated cost \$62.87 / month (Capacity calculator)

Auto Scaling

☒ Read capacity ☐ Write capacity

Target utilization 70 %

Minimum provisioned capacity 300 units

Maximum provisioned capacity 600 units

☒ Apply same settings to global secondary indexes

IAM Role Authorize DynamoDB to scale capacity using the following role:

☒ DynamoDB AutoScaling Service Linked Role

Role Name* AWSServiceRoleForApplicationAutoScaling_DynamoDB

Save Cancel

Auto Scaling

300 → 640 provisioned

70% of 640 ~ 450

0x12A8	236294	{ name:"Sara", city:"Tampa", ...}
0x3391	445104	{ name:"James", city:"Miami", ...}
0x6134	333363	{ name:"Betty", city:"Madison", ...}

150 RCU admitted

0x9531	145783	{ name:"Bob", city:"London", ...}
0xB082	643145	{ name:"Val", city:"Seattle", ...}

150 RCU admitted

0xEA8A	723342	{ name:"Jeff", city:"Toledo", ...}
0xF355	523422	{ name:"Alex", city:"London", ...}

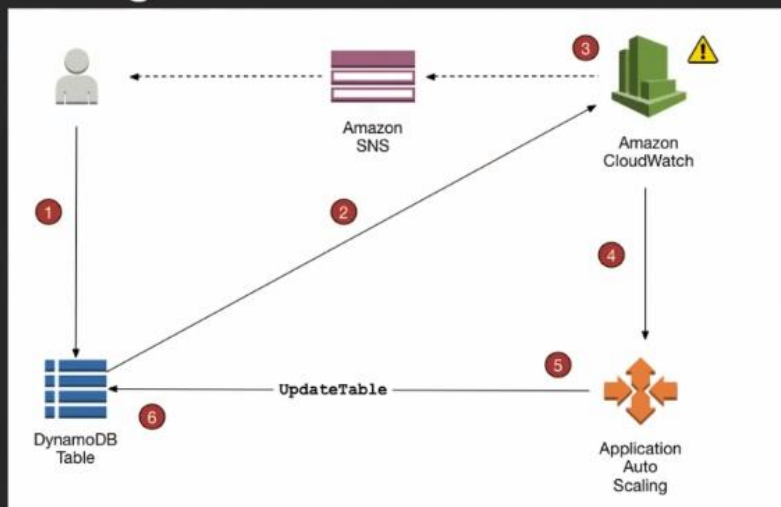
150 RCU admitted

aws
re:Invent

© 2018, Amazon Web Services, Inc. or its affiliates. All rights reserved.

aws

Auto Scaling



aws
re:Invent

© 2018, Amazon Web Services, Inc. or its affiliates. All rights reserved.

aws

Auto-Scaling sets 2 alarms per table in CloudWatch that you can use for reads, for the provisioned value and for the target provisioned value.

Auto Scaling

450 → 43 provisioned

70% of 43 ~ 30

0x12A8	236294	{ name:"Sara", city:"Tampa", ...}
0x3391	445104	{ name:"James", city:"Miami", ...}
0x6134	333363	{ name:"Betty", city:"Madison", ...}

10 RCU Consumed

0x9531	145783	{ name:"Bob", city:"London", ...}
0xB082	643145	{ name:"Val", city:"Seattle", ...}

10 RCU Consumed

0xEA8A	723342	{ name:"Jeff", city:"Toledo", ...}
0xF355	523422	{ name:"Alex", city:"London", ...}

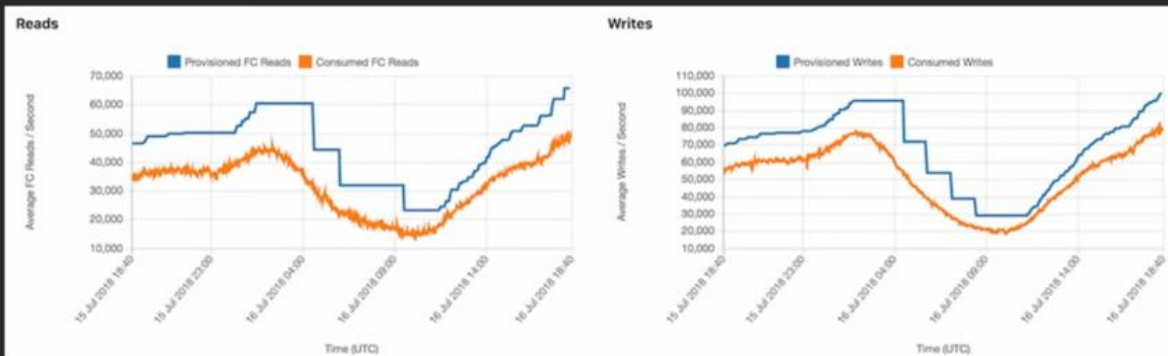
10 RCU Consumed

aws
re:Invent

© 2018, Amazon Web Services, Inc. or its affiliates. All rights reserved.



Actual Auto-scaled Table



aws
re:Invent

© 2018, Amazon Web Services, Inc. or its affiliates. All rights reserved.



Provisioning Recap

- Balanced Usage
 - Imbalance in Time
 - Imbalance in Key Space
 - Changing Workloads
- Bursting
Adaptive Capacity
Auto Scaling

We will continue to iterate

Backup And Restore

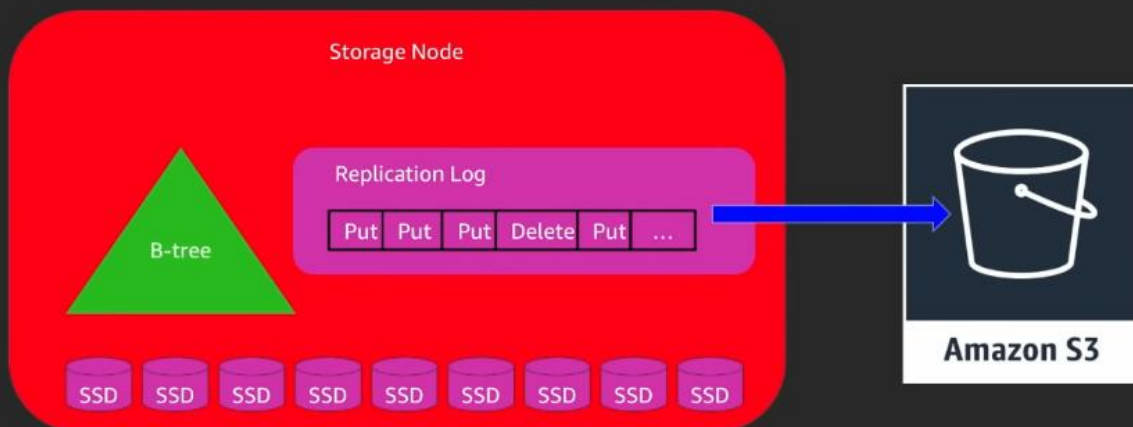
Backup and Restore

- Point in Time
- On demand Backups

Q: Where to durably store backups?

A: Amazon Simple Storage Service (Amazon S3)

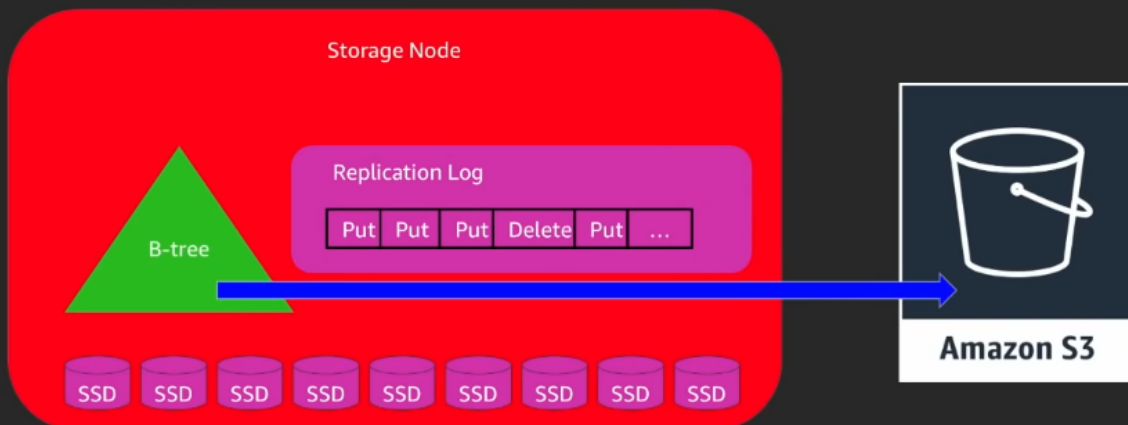
Replication Log to Amazon S3



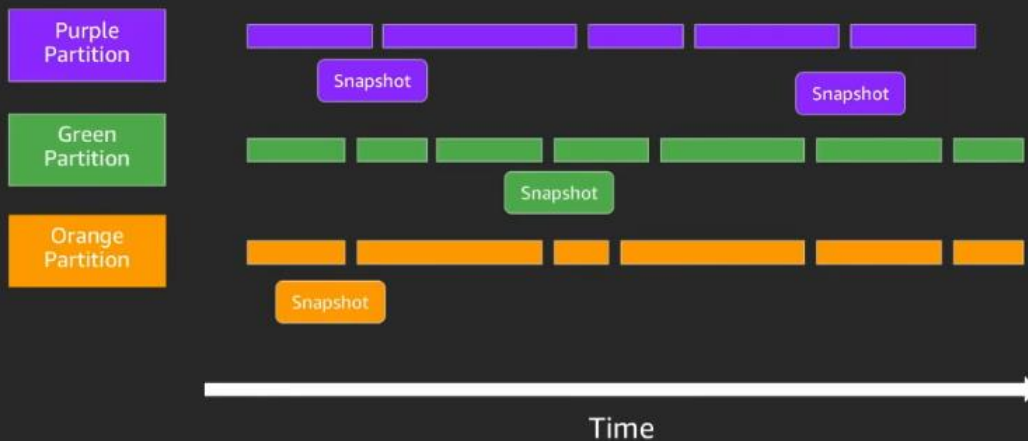
Replication Log to Amazon S3



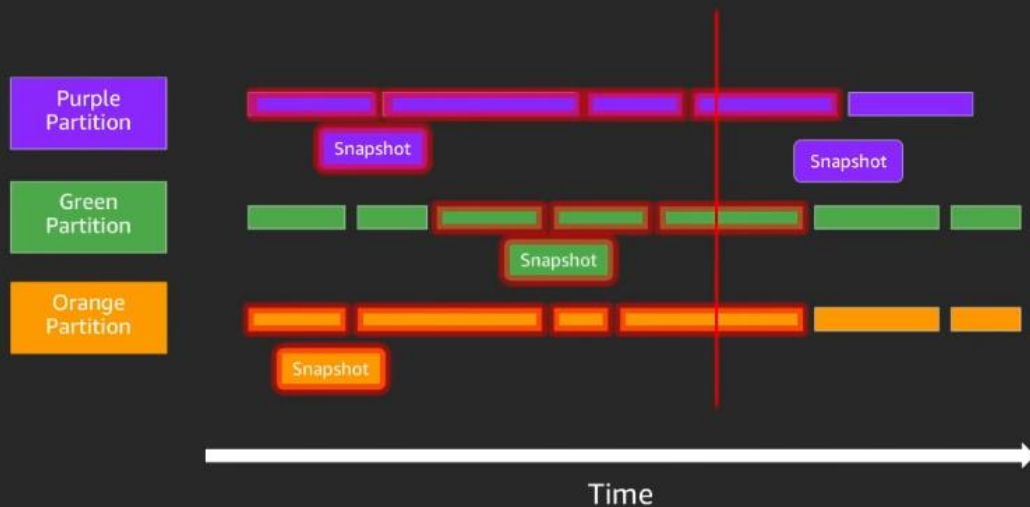
"Snapshot" the B-tree to Amazon S3



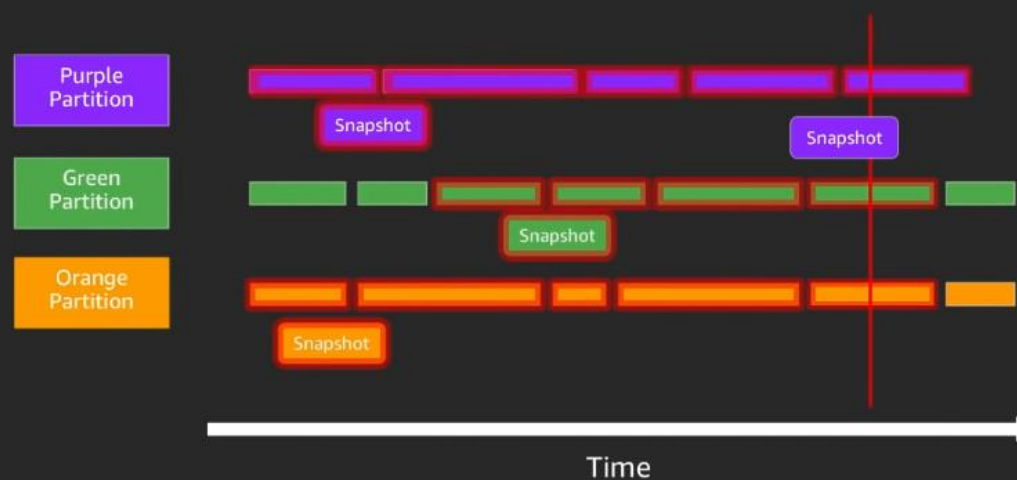
"Snapshot" the B-tree to Amazon S3



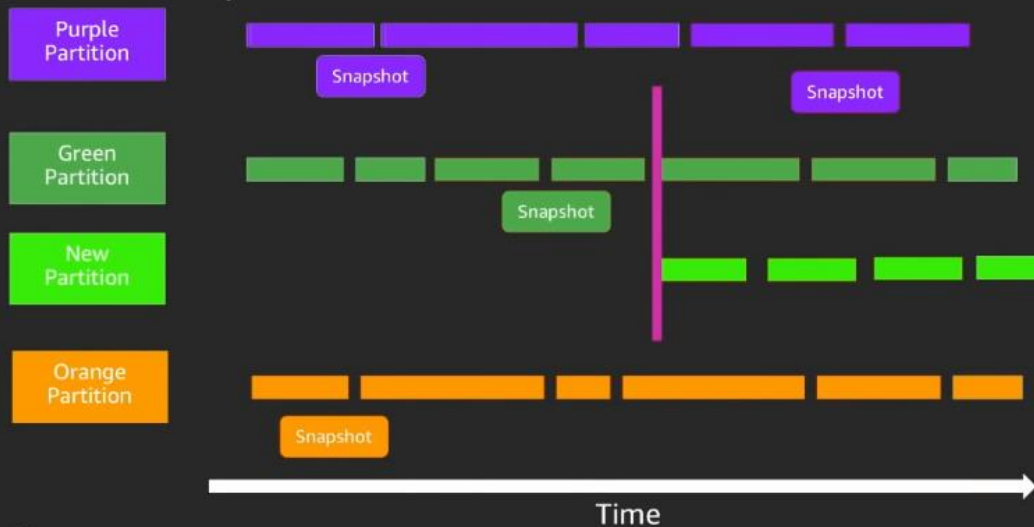
Restore to a Point in Time



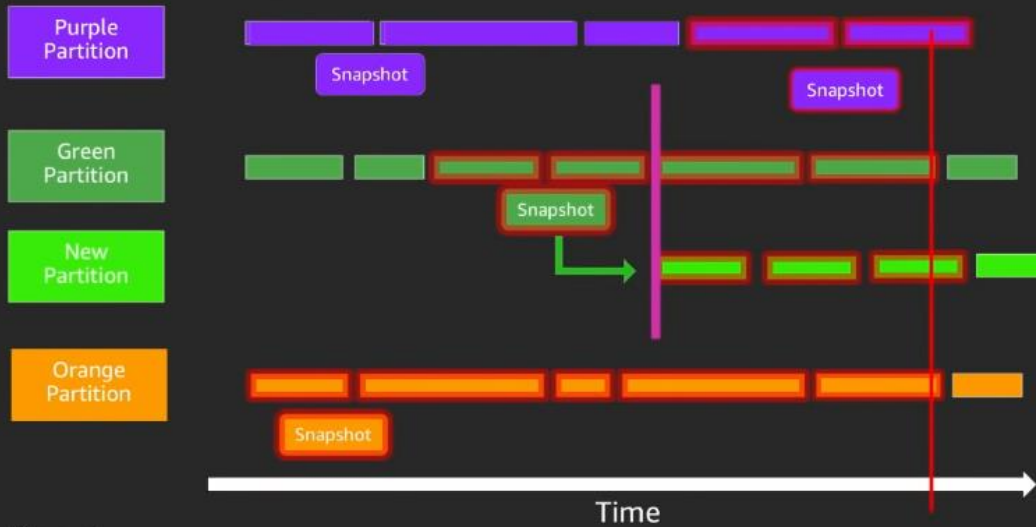
Restore to a Point in Time



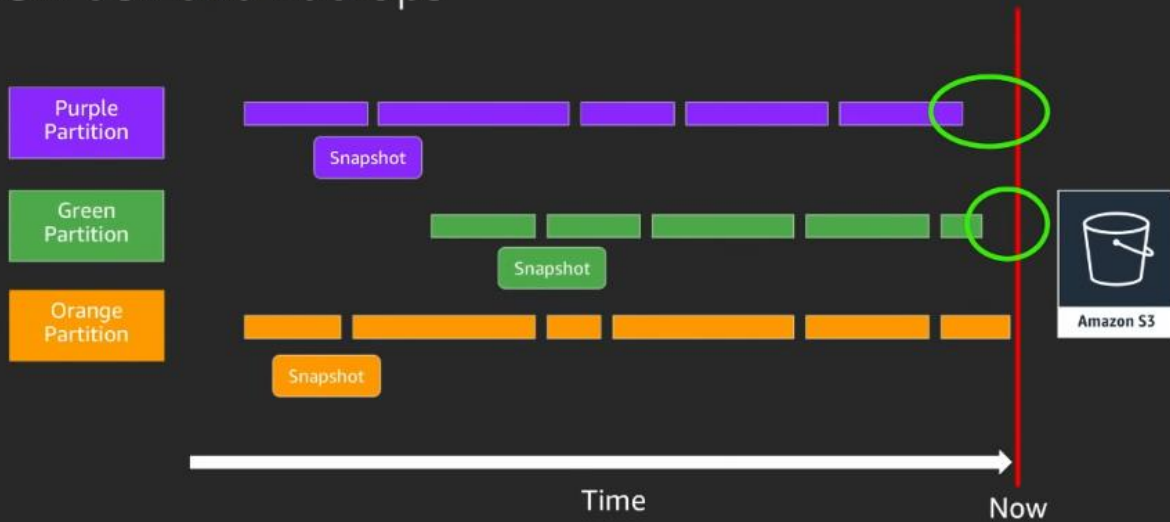
Partition Splits



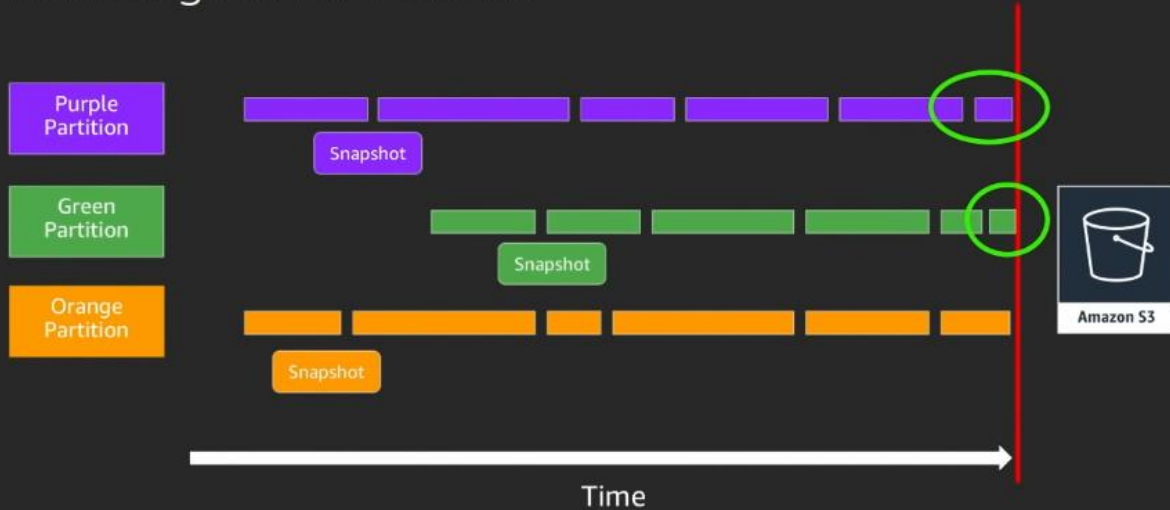
Restore to a Point in Time



On-demand Backups



Write Logs to Amazon S3



On-demand backup

aws Services Resource Groups

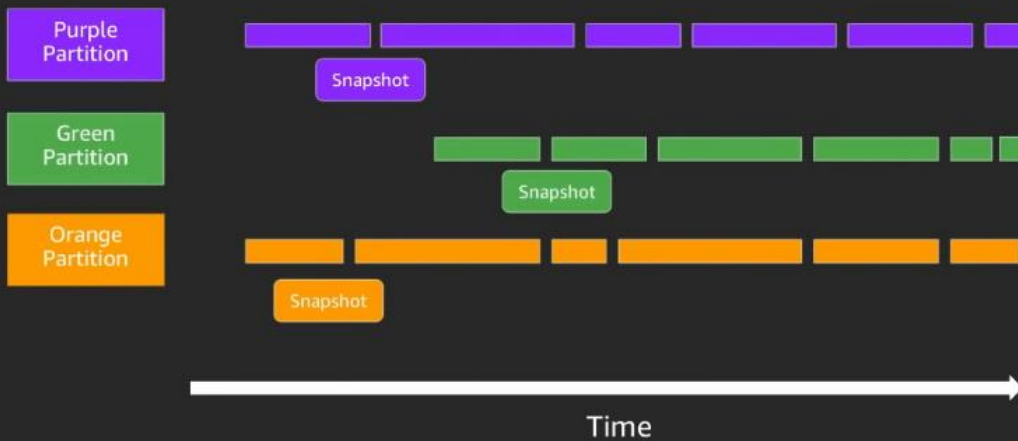
DynamoDB Dashboard Tables Backups Reserved capacity Preferences DAX Dashboard Clusters Subnet groups Parameter groups Events

Create backup Restore backup Delete backup

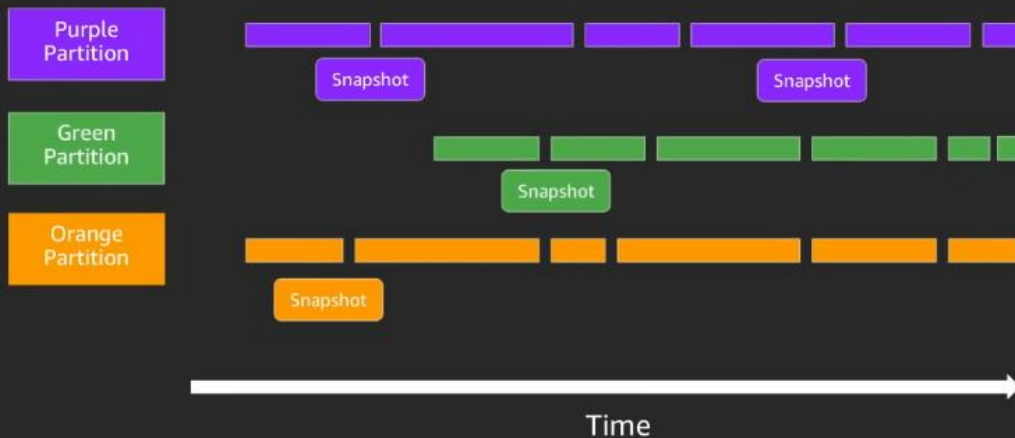
Filter by backup or table name X Time Range Last 30 Days Backup type All backups 1 to 1 of 1 backup

Backup name	Status	Table	Creation time	Size	Backup type	Expiration date	Backup ARN
SaturdayBackup	Available	customer	November 24, 2018 at 5:46:39 PM UTC-8	59.00 bytes	User	-	arn:aws:dynamodb:us-west-2:241781061306:table/cu

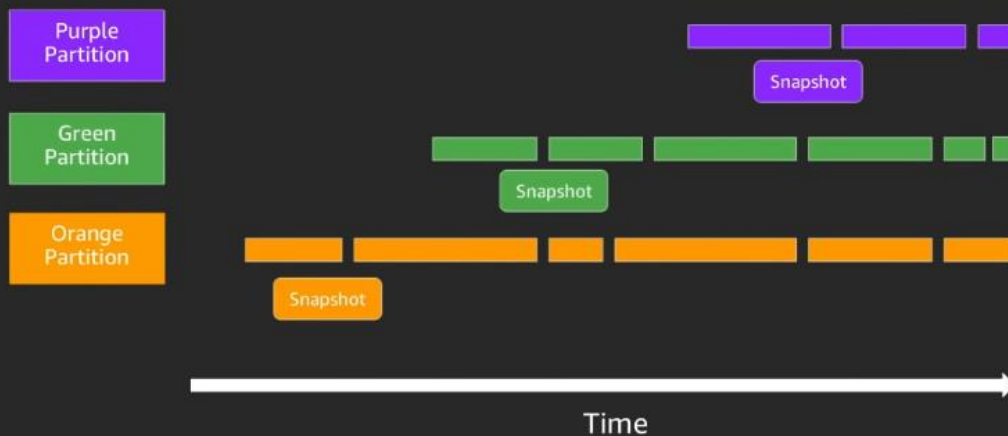
Without PITR enabled



Without PITR enabled



Without PITR enabled



We can delete older backups

With PITR enabled

Overview Items Metrics Alarms Capacity Indexes Global Tables **Backups** Triggers Access control Tags

Point-in-time Recovery

DynamoDB maintains continuous backups of your table for the last 35 days. [Learn more](#)

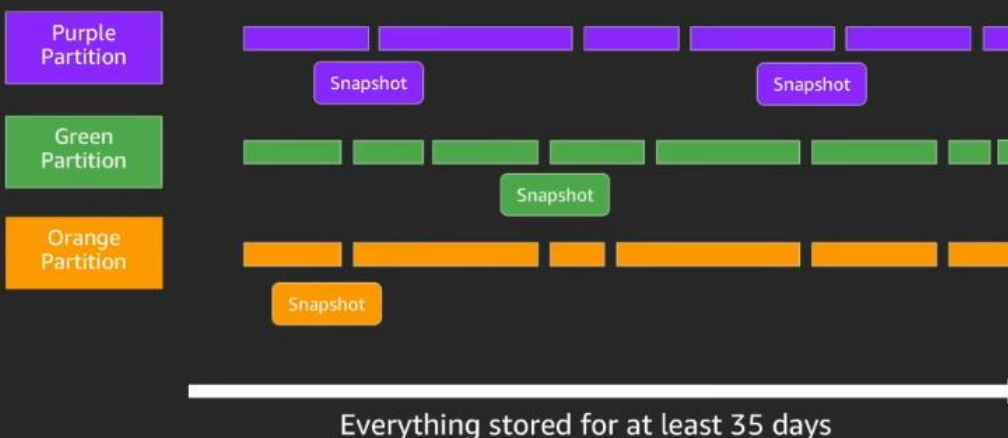
Status	ENABLED Disable
Earliest restore date	November 24, 2018 at 5:59:04 PM UTC-8
Latest restore date	November 24, 2018 at 6:07:08 PM UTC-8

[Restore to point-in-time](#)

On-Demand Backup and Restore

You can create and restore a complete backup of your DynamoDB table data and its settings at any time. [Learn more](#)

With PITR enabled



We now have to maintain this data for at least 35 days, that is why we charge you if you turn PITR on.

DynamoDB Streams

DynamoDB Streams



- All table mutations (Put, Update, Delete)
- No duplicates
- In Order (By Key)
- New and Old Item Image Available

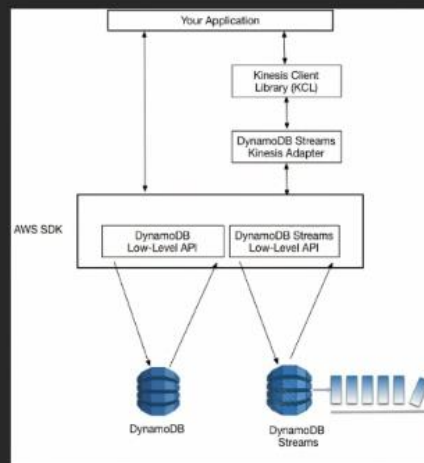
This is a way of getting every mutation to your table.

DynamoDB Streams



Shards
Kinesis Client Library
Records
Stream Checkpointing

Reading from the stream

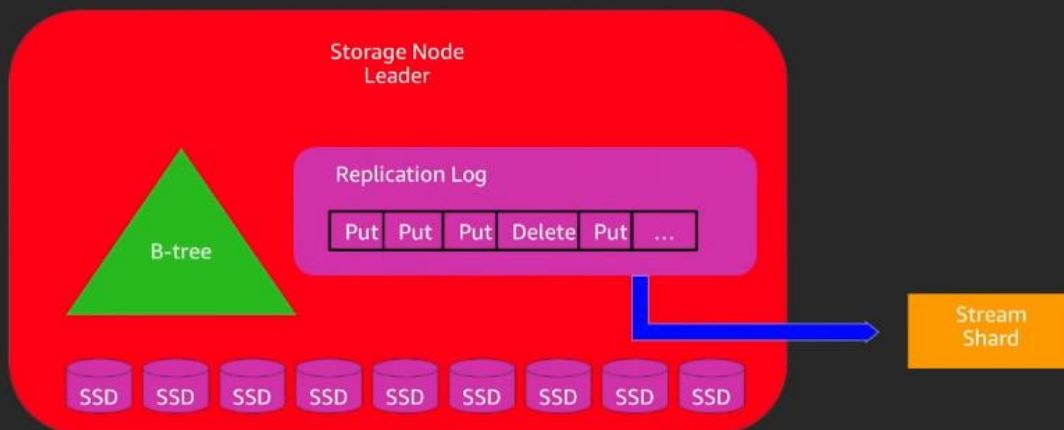


aws
re:Invent

© 2018, Amazon Web Services, Inc. or its affiliates. All rights reserved.



Writing to DynamoDB Streams



Global Tables

Global Tables

ExampleGlobalTable Close

Overview Items Metrics Alarms Capacity Indexes **Global Tables** Backups Triggers Access control Tags

Global Tables enable you to use DynamoDB as a fully-managed, multi-region, multi-master database. [Learn more](#)

IAM role `AWSServiceRoleForDynamoDBReplication`
Automatically created on your behalf.

Global Table regions

Create a replica table in another region. [Learn more](#)

[Add region](#) [Remove region](#)

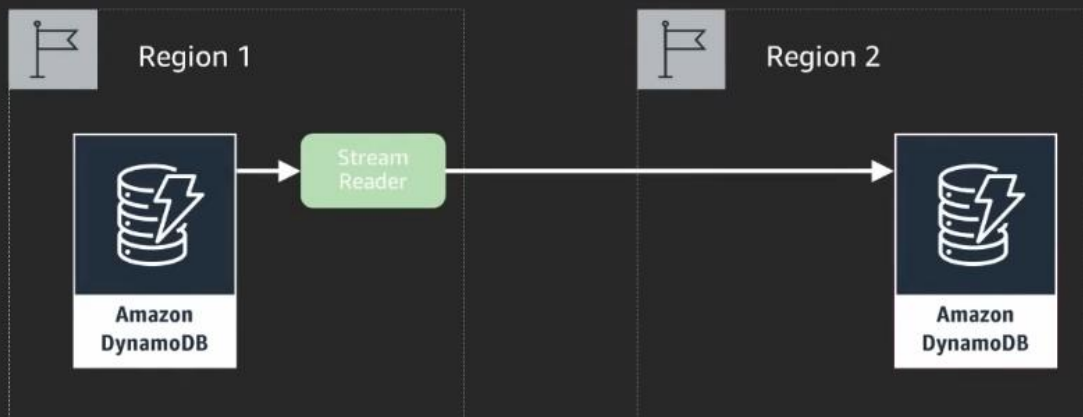
Region Name	Status	Read capacity units	Write capacity units	Auto Scaling	Endpoint
Asia Pacific (Tokyo)	Active	5	5	READ_AND_WRITE	dynamodb.ap-northeast-1.amazonaws.com
EU (London)	Active	5	5	READ_AND_WRITE	dynamodb.eu-west-2.amazonaws.com
US West (Oregon)	Active	5	5	READ_AND_WRITE	dynamodb.us-west-2.amazonaws.com

[View all CloudWatch metrics](#)

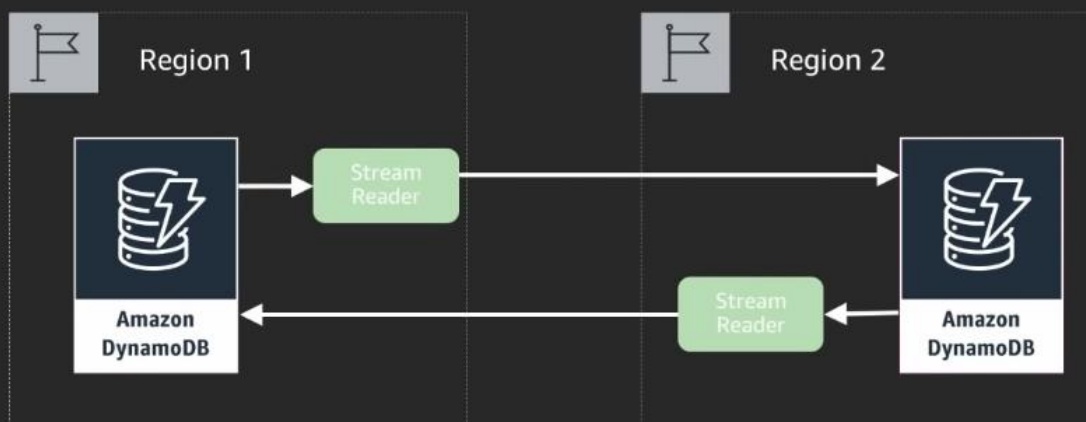
Time Range: Last Hour

There is an IAM role associated with using Global Tables with DynamoDB

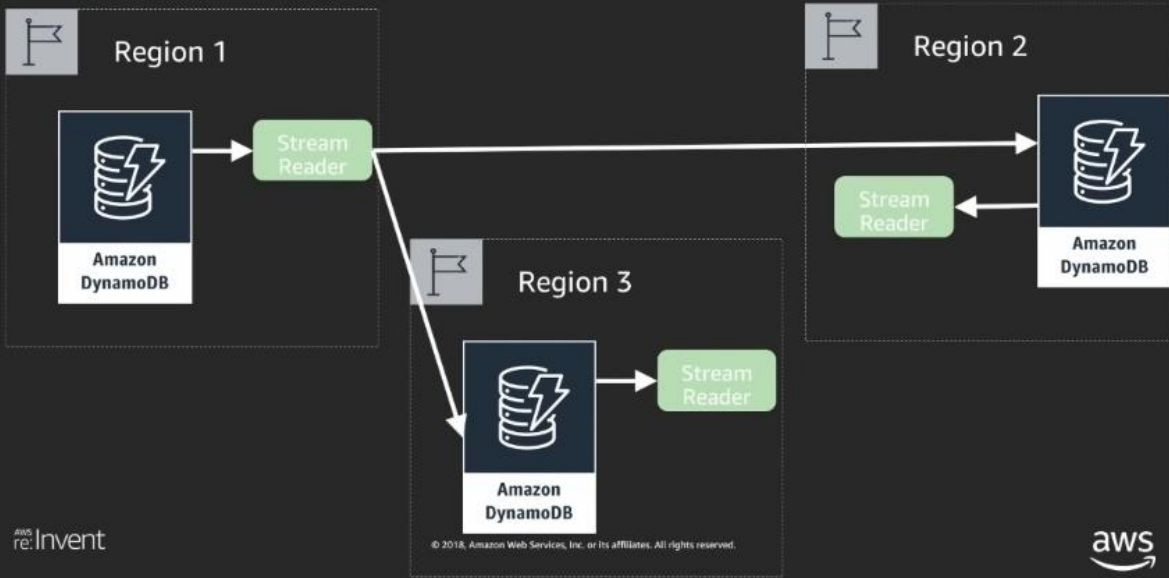
High Level Architecture



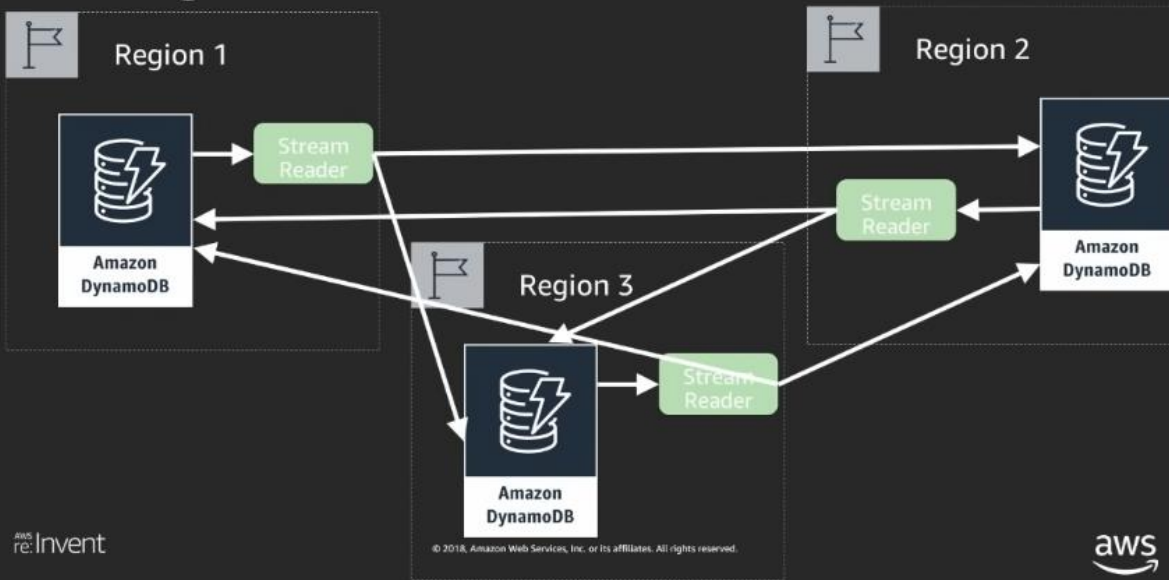
Multi-master replication



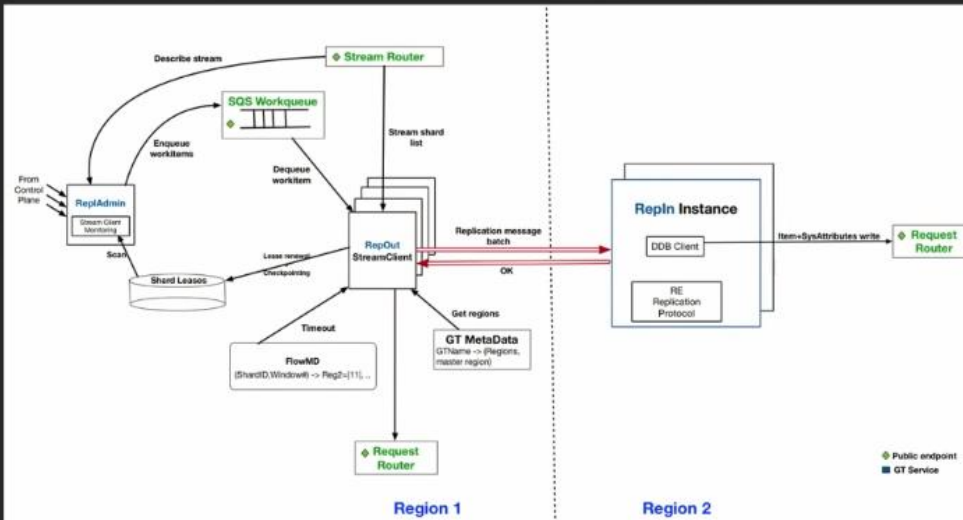
Multi-region replication



Multi-region



"Real" Architecture



aws re:Invent

© 2018, Amazon Web Services, Inc. or its affiliates. All rights reserved.



Initial PutItem

ExampleGlobalTable Close

Overview Items Metrics Alarms Capacity Indexes Global Tables Backups Triggers Access control Tags

Create item Actions

Scan: [Table] ExampleGlobalTable: ExampleKey Viewing 1 to 1 items

Scan [Table] ExampleGlobalTable: ExampleKey

Add filter

Start search

ExampleKey	ExampleValue
key	value

After RepOut

ExampleGlobalTable Close

Overview Items Metrics Alarms Capacity Indexes Global Tables Backups Triggers Access control Tags

Create item Actions

Scan: [Table] ExampleGlobalTable: ExampleKey Viewing 1 to 1 items

Scan [Table] ExampleGlobalTable: ExampleKey

Add filter

Start search

ExampleKey	ExampleValue	aws:rep:deleting	aws:rep:updating	aws:rep:updatetime
key	value	false	us-west-2	1543160171.363001

Conflict Resolution

Last Writer wins.

- aws:rep:deleting
- aws:rep:updatetime
- aws:rep:updateregion

Areas Covered

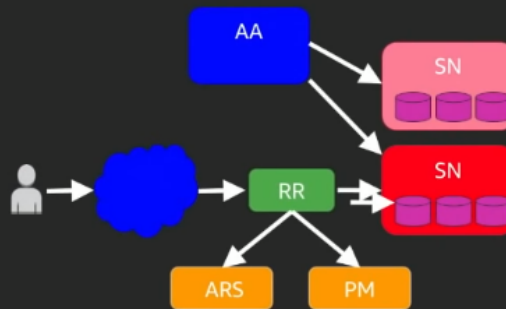
GetItem / PutItem

Auto Scaling

Backup Restore

Streams

Global Tables



Areas Not Covered

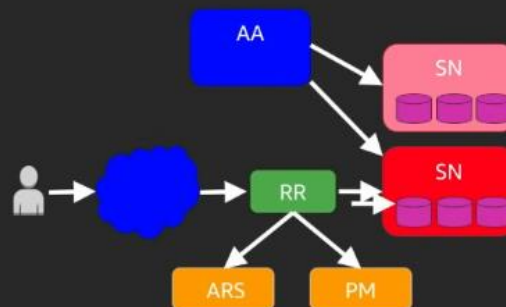
Fleet management

Metering

Monitoring and Alarming

Capacity Planning

...



Thank you!

James Sorenson a.k.a. Jaso
jaso@amazon.com

aws
re:Invent

© 2018, Amazon Web Services, Inc. or its affiliates. All rights reserved.

