Masterclass

Amazon DynamoDB

Ian Massingham – Technical Evangelist
@IanMmmm

Masterclass

A technical deep dive beyond the basics

Help educate you on how to get the best from AWS technologies

Show you how things work and how to get things done

Broaden your knowledge in ~45 mins

Amazon DynamoDB

A fast, fully managed NoSQL database service

Simple & cost effective to store and retrieve any amount of data

Reliable throughput & single-digit millisecond latency

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

ABSTRACT

Reliability at massive scale is one of the biggest challenges we face at Amazon.com, one of the largest e-commerce operations in the world; even the slightest outage has significant financial consequences and impacts customer trust. The Amazon.com platform, which provides services for many web sites worldwide, is implemented on top of an infrastructure of tens of thousands of servers and network components located in many datacenters around the world. At this scale, small and large components fail continuously and the way persistent state is managed in the face of these failures drives the reliability and scalability of the software systems.

This paper presents the design and implementation of Dynamo, a highly available key-value storage system that some of Amazon's core services use to provide an "always-on" experience. To achieve this level of availability, Dynamo sacrifices consistency under certain failure scenarios. It makes extensive use of object versioning and application-assisted conflict resolution in a manner that provides a novel interface for developers to use.

Categories and Subject Descriptors

D.4.2 [**Operating Systems**]: Storage Management; D.4.5 [**Operating Systems**]: Reliability; D.4.2 [**Operating Systems**]: Performance;

General Terms

Algorithms, Management, Measurement, Performance, Design, Reliability.

1. INTRODUCTION

Amazon runs a world-wide e-commerce platform that serves tens of millions customers at peak times using tens of thousands of servers located in many data centers around the world. There are strict operational requirements on Amazon's platform in terms of performance, reliability and efficiency, and to support continuous growth the platform needs to be highly scalable. Reliability is one of the most important requirements because even the slightest

One of the lessons our organization has learned from operating Amazon's platform is that the reliability and scalability of a system is dependent on how its application state is managed. Amazon uses a highly decentralized, loosely coupled, service oriented architecture consisting of hundreds of services. In this environment there is a particular need for storage technologies that are always available. For example, customers should be able to view and add items to their shopping cart even if disks are failing, network routes are flapping, or data centers are being destroyed by tornados. Therefore, the service responsible for managing shopping carts requires that it can always write to and read from its data store, and that its data needs to be available across multiple data centers.

Dealing with failures in an infrastructure comprised of millions of components is our standard mode of operation; there are always a small but significant number of server and network components that are failing at any given time. As such Amazon's software systems need to be constructed in a manner that treats failure handling as the normal case without impacting availability or performance.

To meet the reliability and scaling needs, Amazon has developed a number of storage technologies, of which the Amazon Simple Storage Service (also available outside of Amazon and known as Amazon S3), is probably the best known. This paper presents the design and implementation of Dynamo, another highly available and scalable distributed data store built for Amazon's platform. Dynamo is used to manage the state of services that have very high reliability requirements and need tight control over the tradeoffs between availability, consistency, cost-effectiveness and performance. Amazon's platform has a very diverse set of applications with different storage requirements. A select set of applications requires a storage technology that is flexible enough to let application designers configure their data store appropriately based on these tradeoffs to achieve high availability and guaranteed performance in the most cost effective manner.

There are many services on Amazon's platform that only need


Easy Adminstration
Fast, Predictable Performance
DynamoDB
Scalable
Built-in Fault Tolerance

Getting Started ◀

Provisioned Throughput ◀

Data Modeling ◀

Programming DynamoDB ◀

Partitioning ◀

Working with the AWS CLI and SDKs ◀

Reporting & Analysis ◀

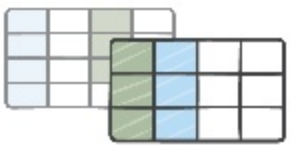
Getting Started

DynamoDB is a managed NoSQL database service

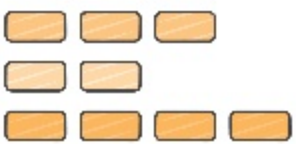
Designed to store and retrieve any amount of data & to serve any level of request traffic

Getting Started

Seamless Scaling



Secondary Indexes



Schema-less

Strong Consistency, Atomic Counters

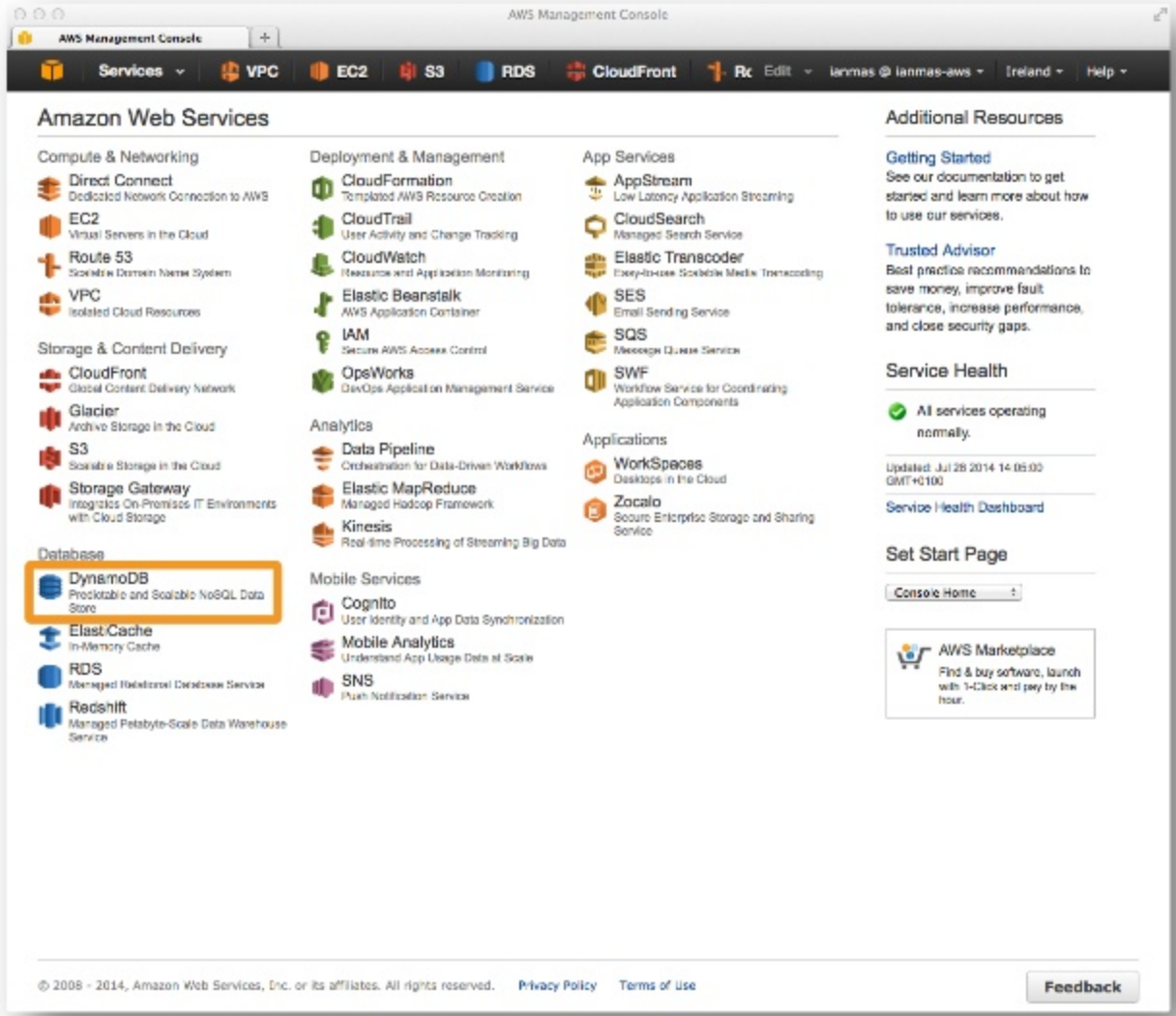
Getting Started

Durability

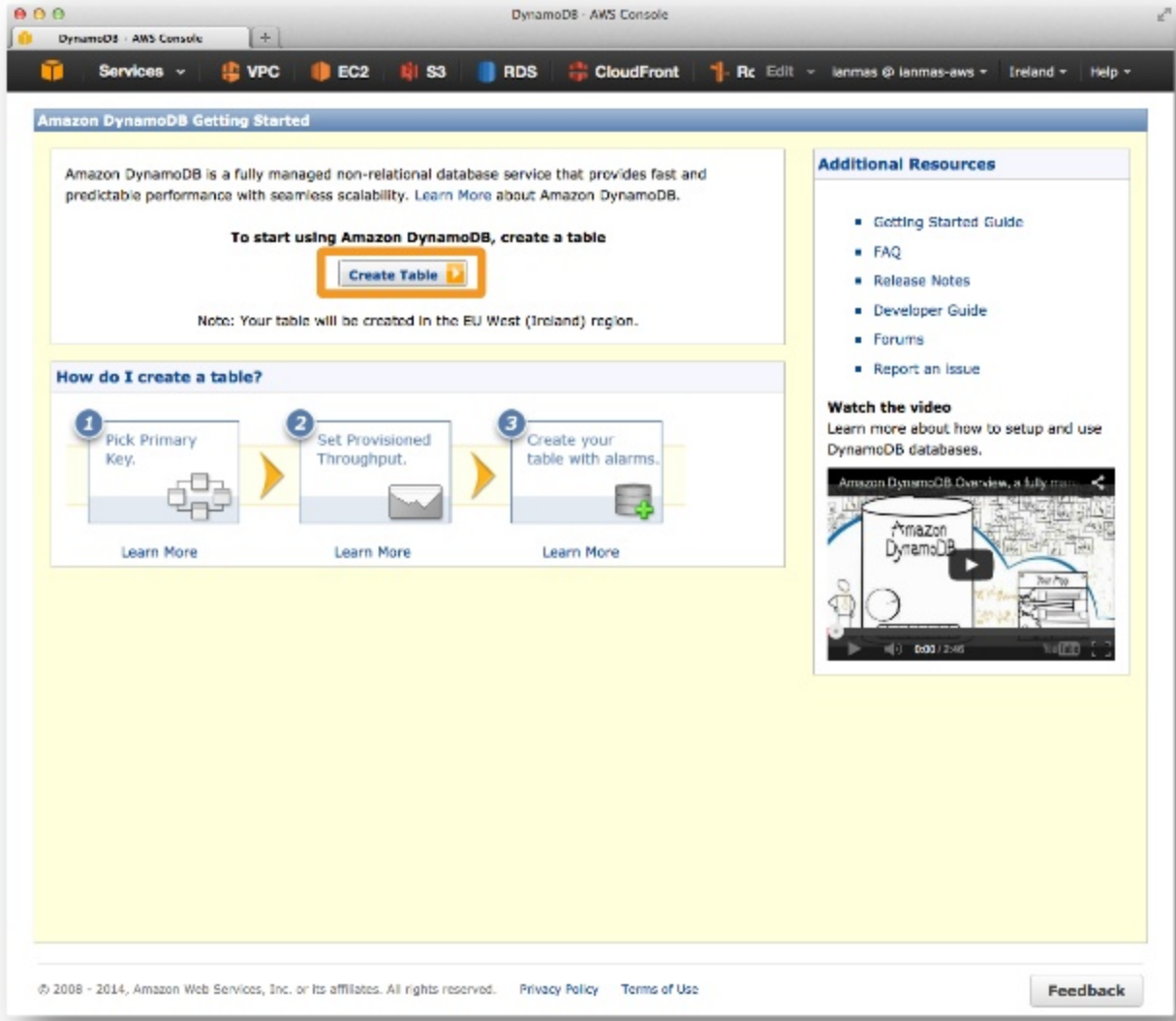
Consistent, disk only writes

Replication across data centers and availability zones

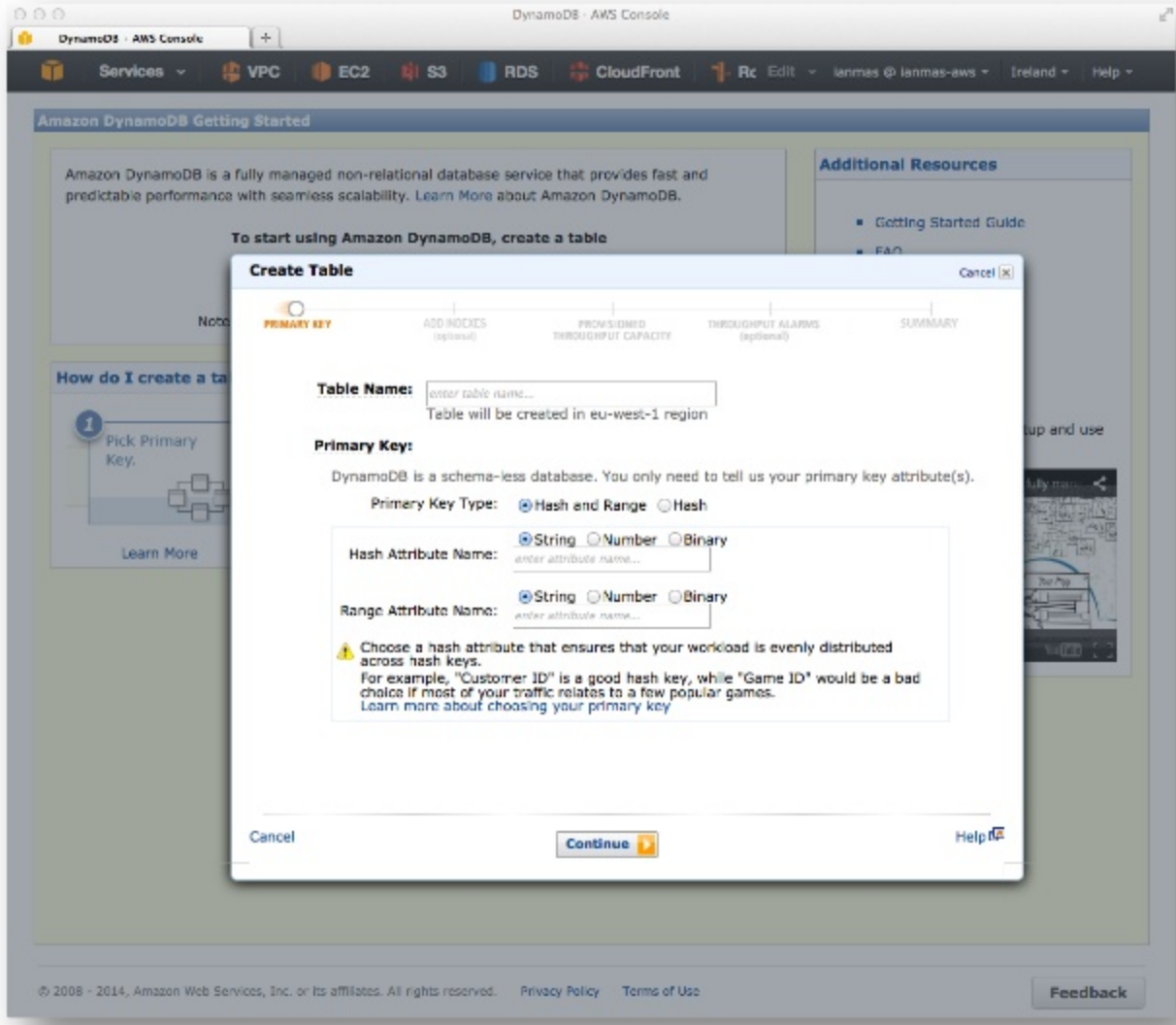
Creating a DynamoDB table via the AWS Console



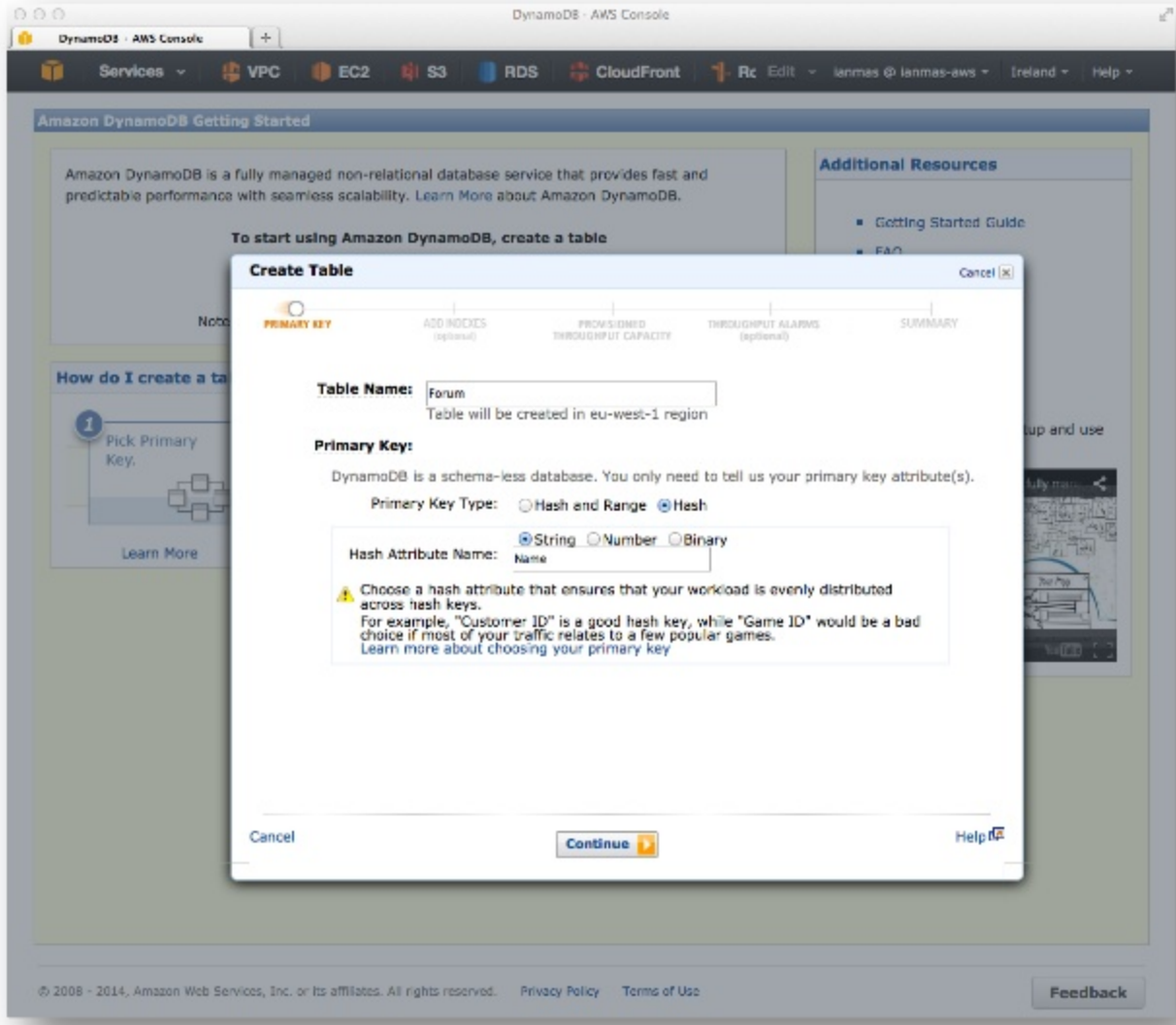
Creating a DynamoDB table via the AWS Console



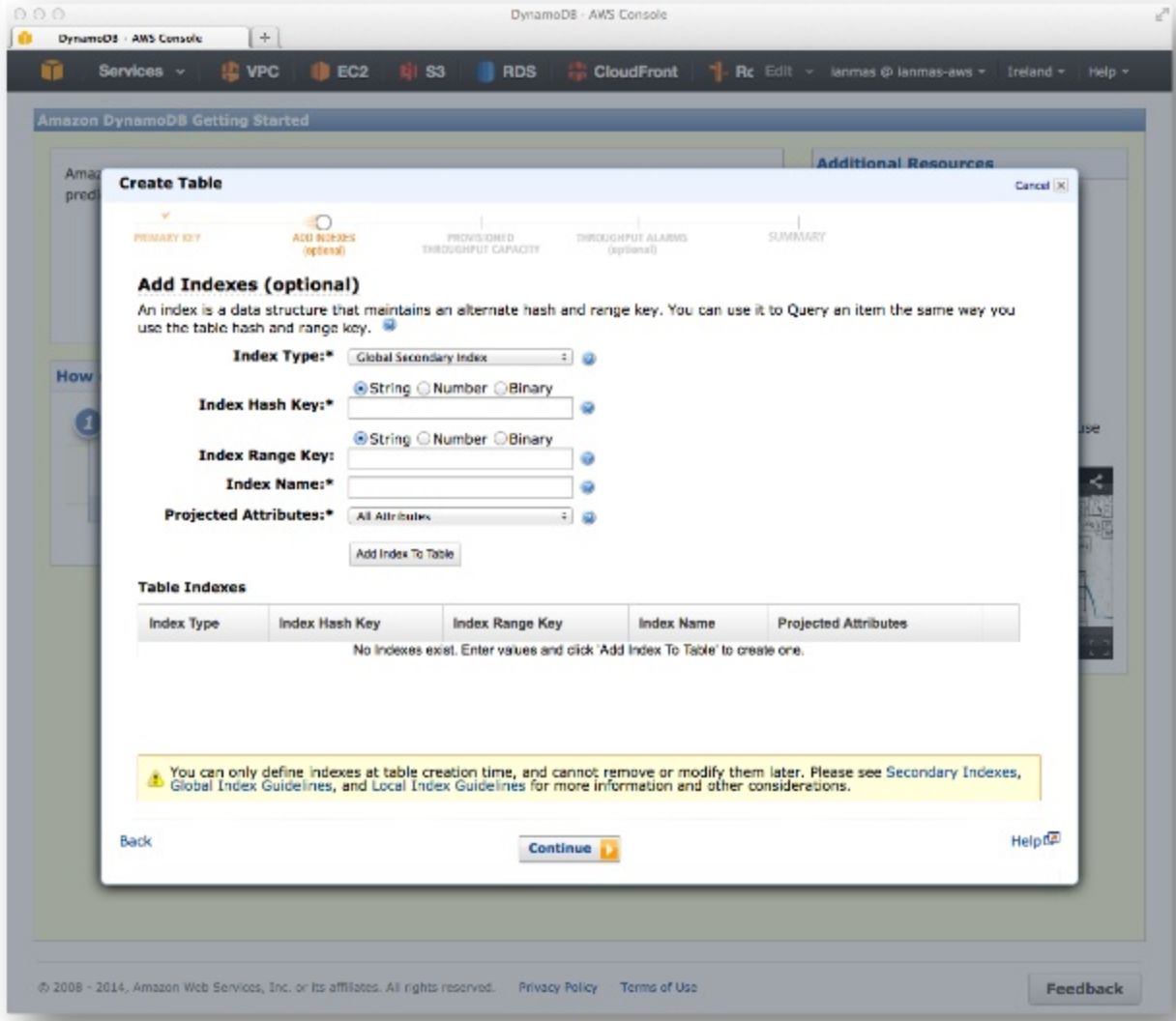
Selecting the Primary Key Type



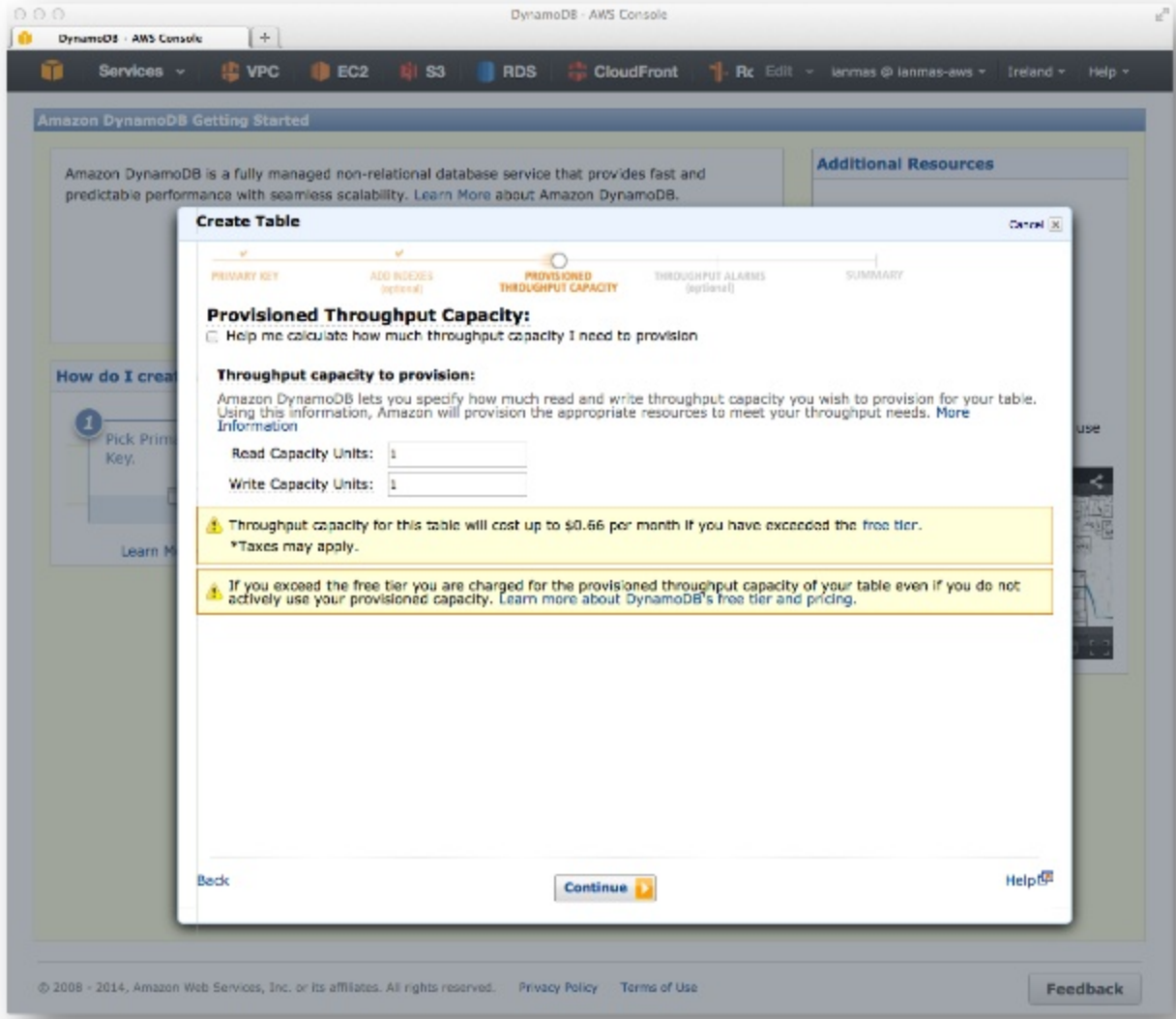
Specifying a Hash Primary Key Type



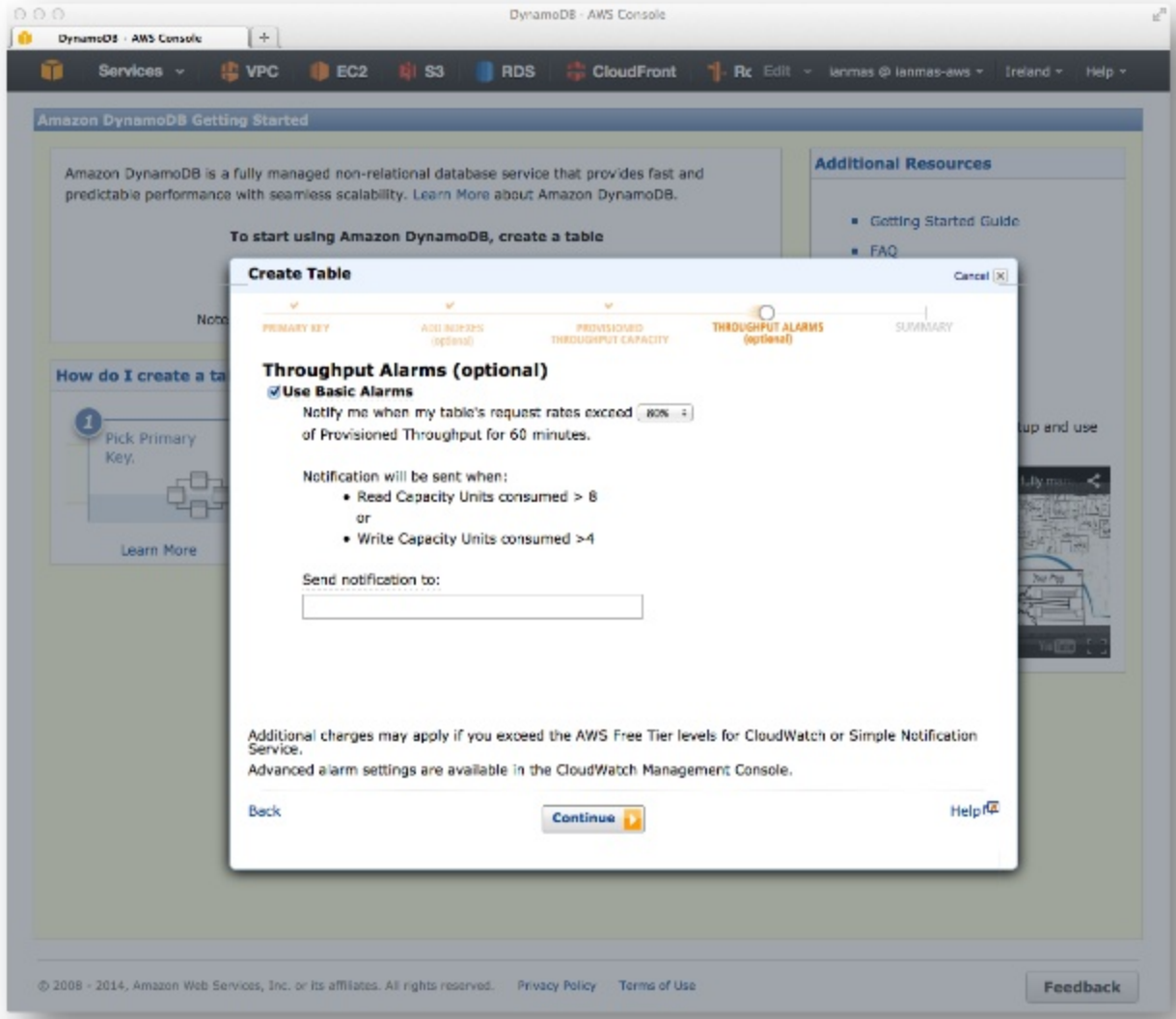
Add optional Indexes to the new table



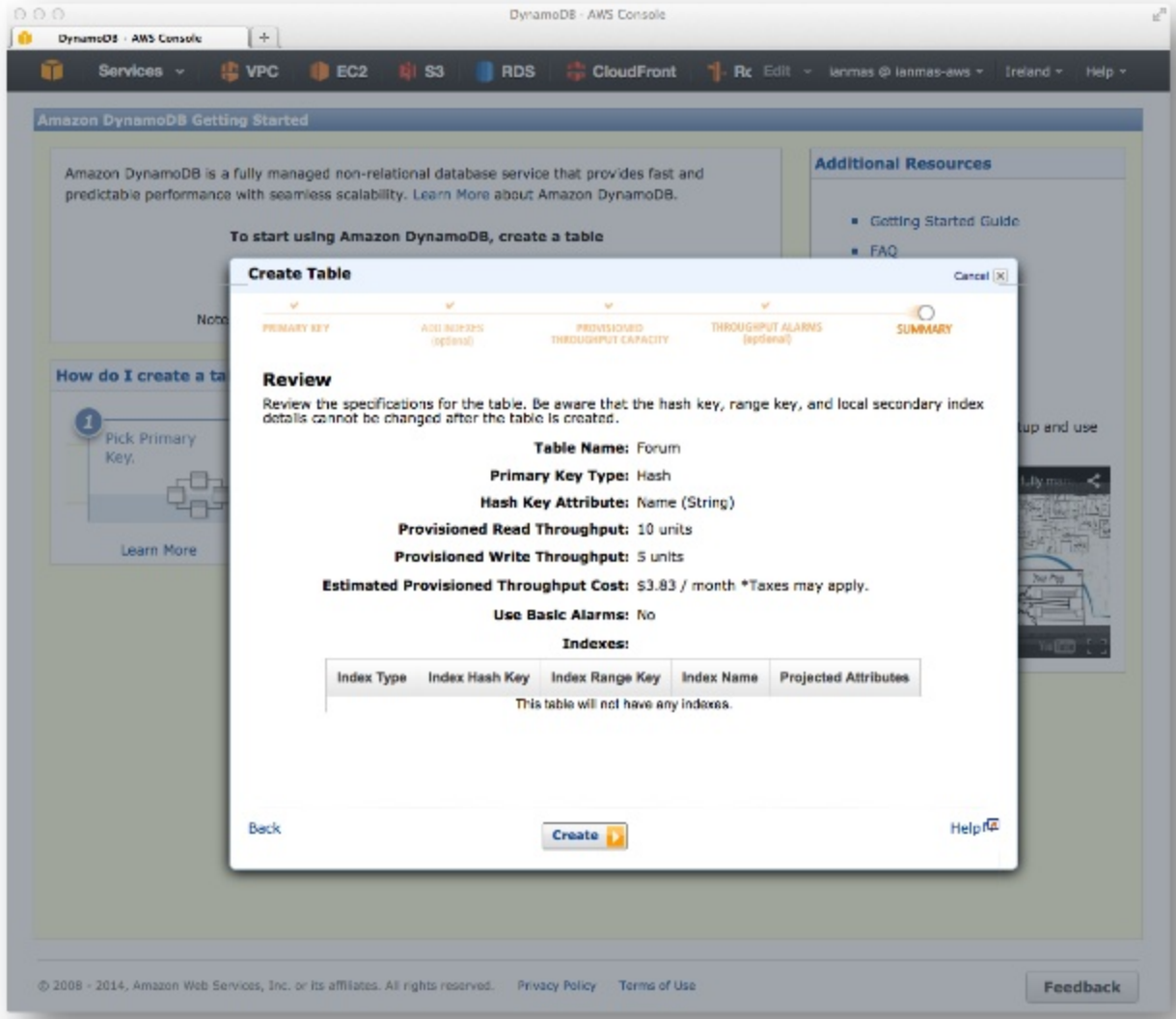
Specify Provisioned Throughput Capacity



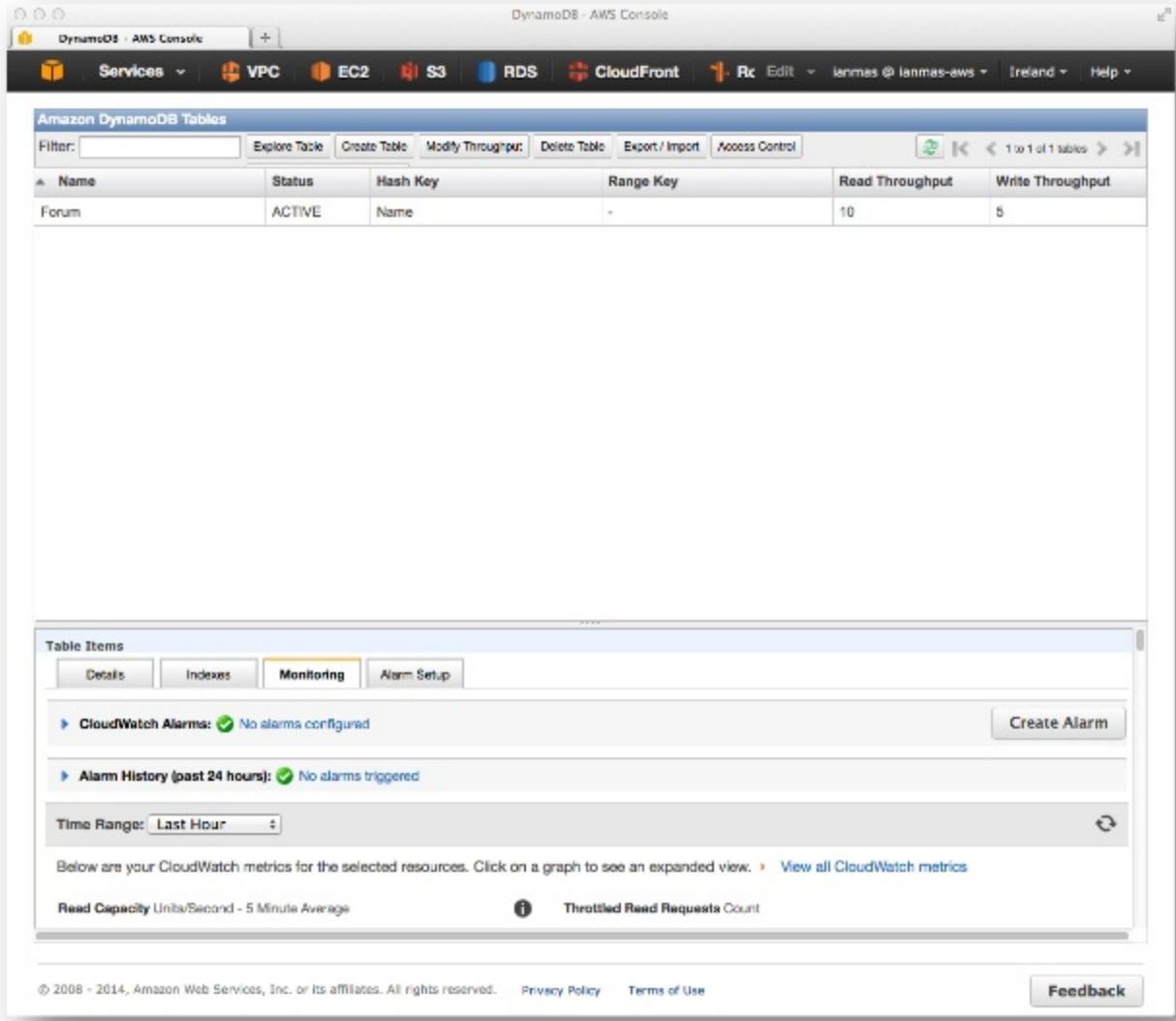
Add optional Throughput Alarms



Review


DynamoDB - AWS Console
Services
VPC
EC2
S3
RDS
CloudFront
Edit
Ireland
Help
Amazon DynamoDB Getting Started
Amazon DynamoDB is a fully managed non-relational database service that provides fast and predictable performance with seamless scalability. Learn More about Amazon DynamoDB.
To start using Amazon DynamoDB, create a table
Additional Resources
Getting Started Guide
FAQ
How do I create a ta
Pick Primary Key.
Learn More
Create Table
Cancel
PRIMARY KEY
ADD INDEXES (optional)
PROVISIONED THROUGHPUT CAPACITY
THROUGHPUT ALARMS (optional)
SUMMARY
Review
Review the specifications for the table. Be aware that the hash key, range key, and local secondary index details cannot be changed after the table is created.
Table Name: Forum
Primary Key Type: Hash
Hash Key Attribute: Name (String)
Provisioned Read Throughput: 10 units
Provisioned Write Throughput: 5 units
Estimated Provisioned Throughput Cost: $3.83 / month *Taxes may apply.
Use Basic Alarms: No
Indexes:
Index Type
Index Hash Key
Index Range Key
Index Name
Projected Attributes
This table will not have any indexes.
Back
Create
Help
© 2008 - 2014, Amazon Web Services, Inc. or its affiliates. All rights reserved.
Privacy Policy
Terms of Use
Feedback

Confirmation that the new table is now active



Getting Started

Ready for use with just three clicks,

but you do have to make two decisions first.

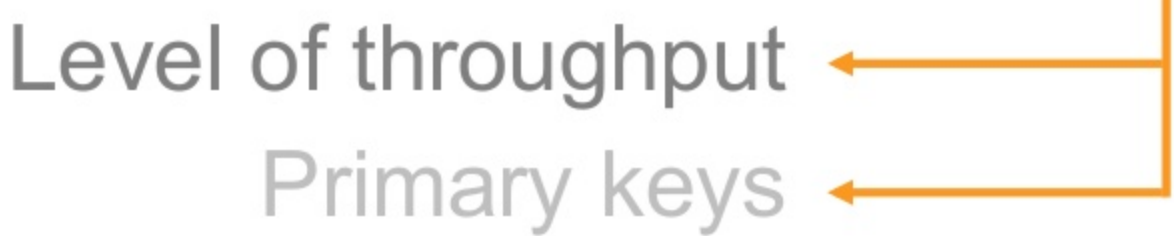
Getting Started Guide: docs.aws.amazon.com/amazondynamodb/latest/developerguide/GettingStartedDynamoDB.html

Ready for use with just three clicks,

but you do have to make two decisions first.

Ready for use with just three clicks,

but you do have to make two decisions first.



Provisioned Throughput

Reserve IOPS for reads and writes

Scale up for down at any time

Pay per capacity unit

Priced per hour of provisioned throughput

Write throughput

Size of item x writes per second

$0.0065 for 10 write units

Consistent writes

Atomic increment and decrement.

Optimistic concurrency control: conditional writes.

Transactions

Item level transactions only

Puts, updates and deletes are ACID

Strong or eventual consistency

Read throughput

Strong or eventual consistency

Read throughput

Provisioned units = (size of item / 4) x reads per second

$0.0065 per hour for 50 units

Strong or eventual consistency

Read throughput

$$\text{Provisioned units} = \frac{(\text{size of item} / 4) \times \text{reads per second}}{2}$$

$0.0065 per hour for 100 units

Strong or eventual consistency

Read throughput

Same latency expectations.

Mix and match at ‘read time’

Provisioned throughput is managed by DynamoDB

Data is partitioned and managed by DynamoDB

Reserved capacity

Up to 53% for 1 year reservation.

Up to 76% for 3 year reservation.

Full Pricing Details : http://aws.amazon.com/dynamodb/pricing/

Authentication

Session based to minimize latency

Uses the Amazon Security Token Service

Handled by AWS SDKs

Integrates with IAM

Monitoring

CloudWatch metrics: latency, consumed read and write throughput, errors & throttling

Data Modeling

Data Modeling

id = 100	date = 2012-05-16-09-00-10	total = 25.00
id = 101	date = 2012-05-15-15-00-11	total = 35.00
id = 101	date = 2012-05-16-12-00-10	total = 100.00
id = 102	date = 2012-03-20-18-23-10	total = 20.00
id = 102	date = 2012-03-20-18-23-10	total = 120.00

Data Modeling

Table

id = 100	date = 2012-05-16-09-00-10	total = 25.00
id = 101	date = 2012-05-15-15-00-11	total = 35.00
id = 101	date = 2012-05-16-12-00-10	total = 100.00
id = 102	date = 2012-03-20-18-23-10	total = 20.00
id = 102	date = 2012-03-20-18-23-10	total = 120.00

Data Modeling

id = 100	date = 2012-05-16-09-00-10	total = 25.00
id = 101	date = 2012-05-15-15-00-11	total = 35.00
id = 101	date = 2012-05-16-12-00-10	total = 100.00
id = 102	date = 2012-03-20-18-23-10	total = 20.00
id = 102	date = 2012-03-20-18-23-10	total = 120.00

Item

Data Modeling

Attribute

id = 100	date = 2012-05-16-09-00-10	total = 25.00
id = 101	date = 2012-05-15-15-00-11	total = 35.00
id = 101	date = 2012-05-16-12-00-10	total = 100.00
id = 102	date = 2012-03-20-18-23-10	total = 20.00
id = 102	date = 2012-03-20-18-23-10	total = 120.00

Where is the schema?

Tables do not require a formal schema

Items are an arbitrarily sized hash

Indexing

Items are indexed by primary and secondary keys

Primary keys can be composite

Secondary keys index on other attributes

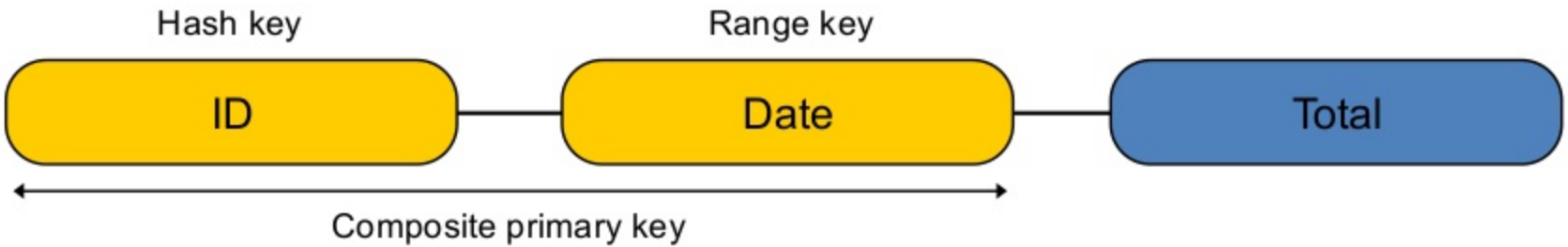
Data Modeling

id = 100	date = 2012-05-16-09-00-10	total = 25.00
id = 101	date = 2012-05-15-15-00-11	total = 35.00
id = 101	date = 2012-05-16-12-00-10	total = 100.00
id = 102	date = 2012-03-20-18-23-10	total = 20.00
id = 102	date = 2012-03-20-18-23-10	total = 120.00

Data Modeling

id = 100	date = 2012-05-16-09-00-10	total = 25.00
id = 101	date = 2012-05-15-15-00-11	total = 35.00
id = 101	date = 2012-05-16-12-00-10	total = 100.00
id = 102	date = 2012-03-20-18-23-10	total = 20.00
id = 102	date = 2012-03-20-18-23-10	total = 120.00

Data Modeling



id = 100	date = 2012-05-16-09-00-10	total = 25.00
id = 101	date = 2012-05-15-15-00-11	total = 35.00
id = 101	date = 2012-05-16-12-00-10	total = 100.00
id = 102	date = 2012-03-20-18-23-10	total = 20.00
id = 102	date = 2012-03-20-18-23-10	total = 120.00

Data Modeling

id = 100	date = 2012-05-16-09-00-10	total = 25.00
id = 101	date = 2012-05-15-15-00-11	total = 35.00
id = 101	date = 2012-05-16-12-00-10	total = 100.00
id = 102	date = 2012-03-20-18-23-10	total = 20.00
id = 102	date = 2012-03-20-18-23-10	total = 120.00

Programming DynamoDB

CreateTable

UpdateTable

DeleteTable

DescribeTable

ListTables

Query

Scan

PutItem

GetItem

UpdateItem

DeleteItem

BatchGetItem

BatchWriteItem

CreateTable

UpdateTable

DeleteTable

DescribeTable

ListTables

Query

Scan

PutItem

GetItem

UpdateItem

DeleteItem

BatchGetItem

BatchWriteItem

CreateTable

UpdateTable

DeleteTable

DescribeTable

ListTables

Query

Scan

PutItem

GetItem

UpdateItem

DeleteItem

BatchGetItem

BatchWriteItem

Conditional updates

PutItem, UpdateItem, DeleteItem can take optional conditions for operation.

UpdateItem performs atomic increments.

One API call, multiple items

BatchGet returns multiple items by key.

BatchWrite performs up to 25 put or delete operations.

Throughput is measured by IO, not API calls.

CreateTable

UpdateTable

DeleteTable

DescribeTable

ListTables

Query

Scan

PutItem

GetItem

UpdateItem

DeleteItem

BatchGetItem

BatchWriteItem

Query vs Scan

Query for Composite Key queries.

Scan for full table scans, exports.

Both support pages and limits.

Maximum response is 1Mb in size.

Query patterns

Retrieve all items by **hash key**

Range key conditions: ==, <, >, >=, <=, begins with, between

Counts, Top and bottom n values, Paged responses

EXAMPLE 1:

Relationships & Secondary Indexes

Social Network

hash + range schemas

Social Network

- Store info about users
- Query for a user’s friends

Social Network

Friends Table

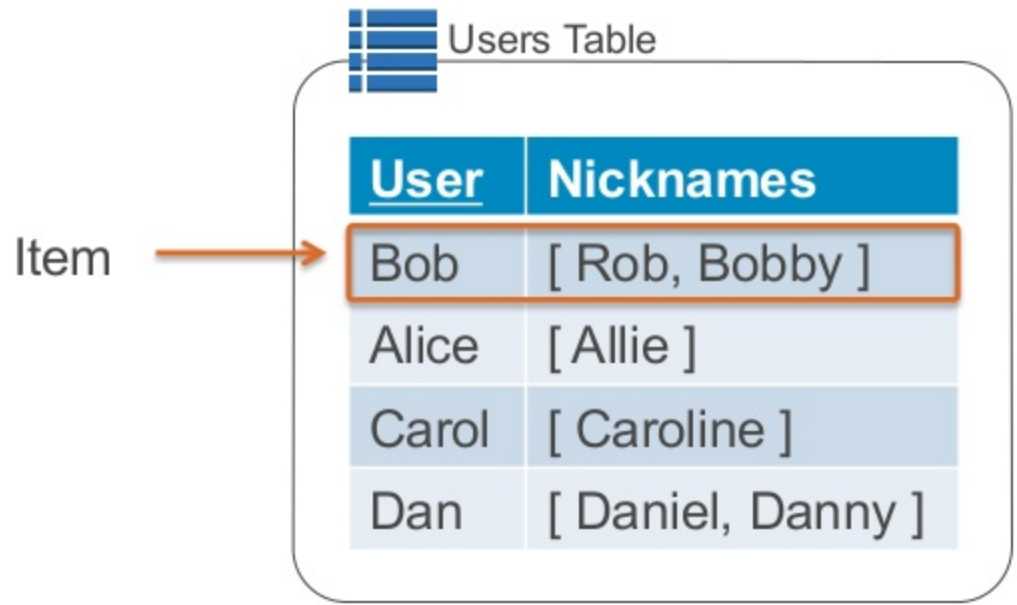
Users Table

Social Network

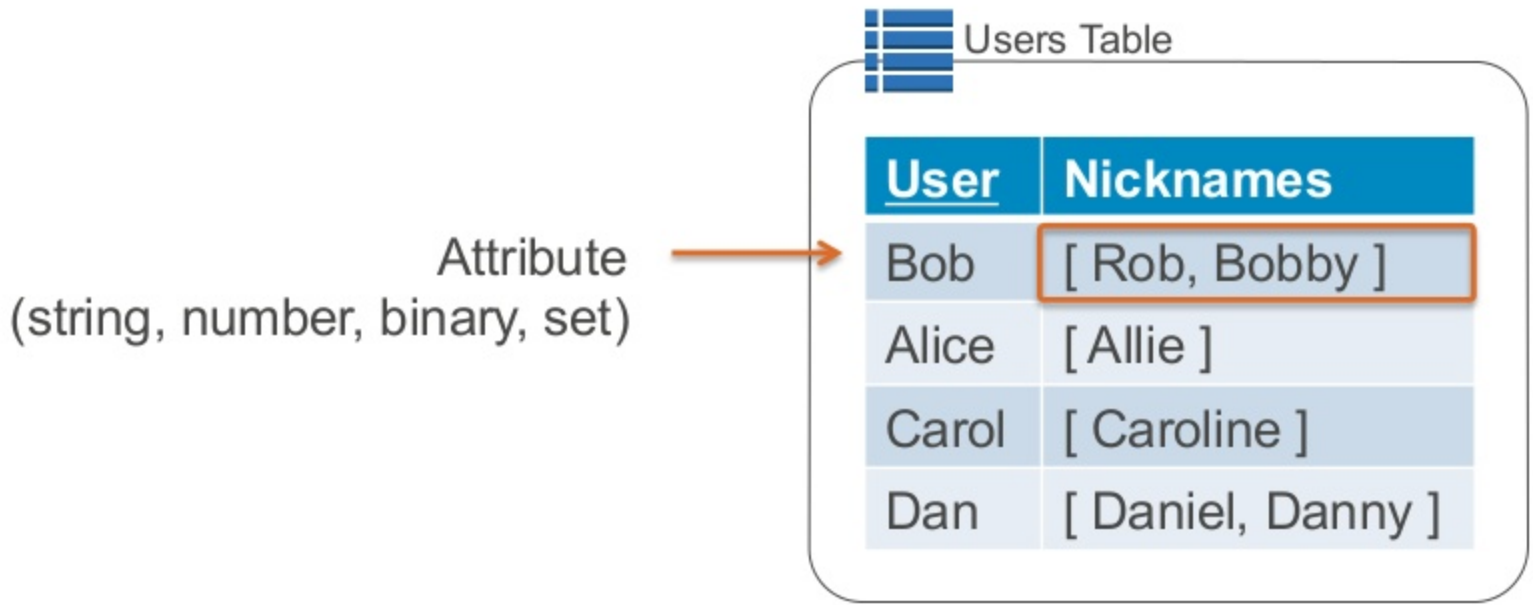
Users Table

User	Nicknames
Bob	[Rob, Bobby]
Alice	[Allie]
Carol	[Caroline]
Dan	[Daniel, Danny]

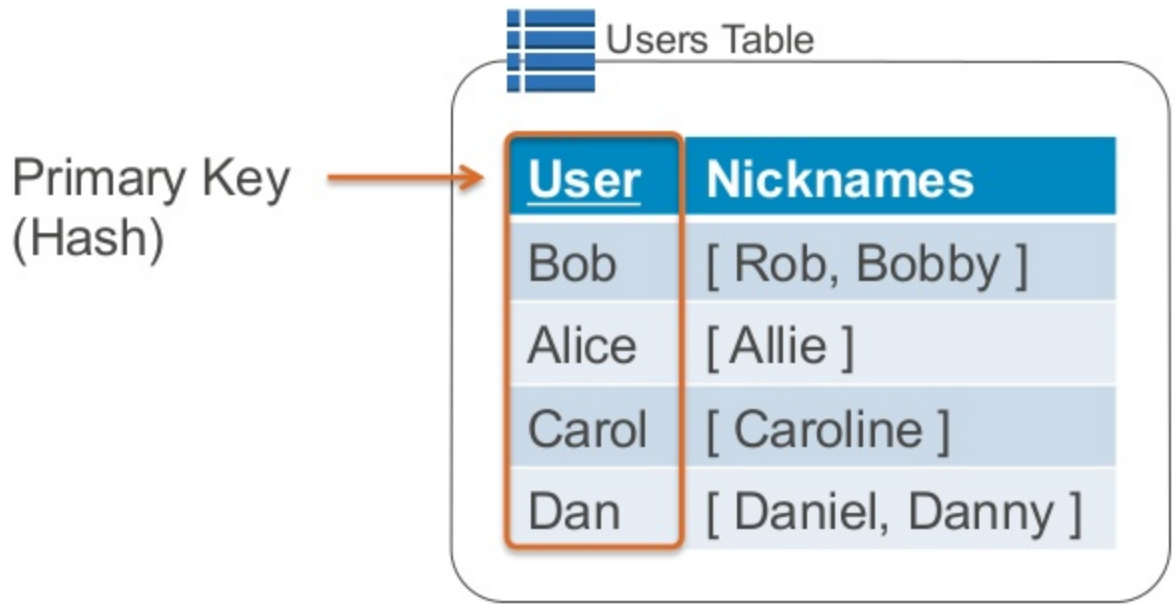
Social Network



Social Network



Social Network



Social Network

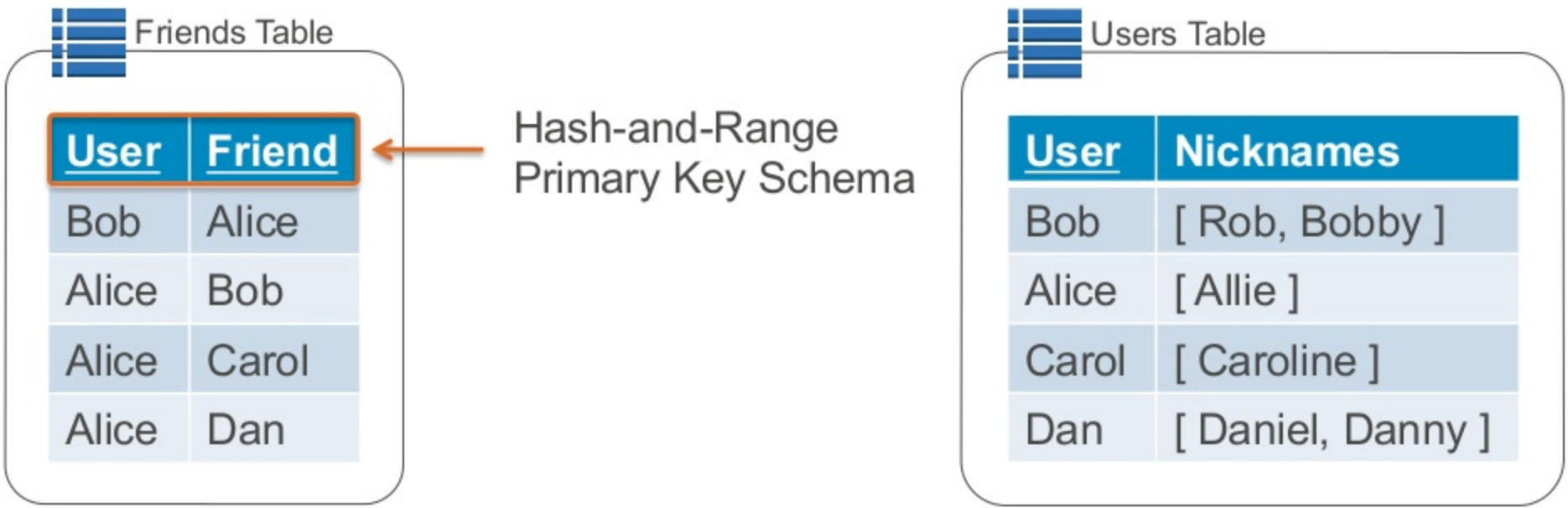
Friends Table

User	Friend
Bob	Alice
Alice	Bob
Alice	Carol
Alice	Dan

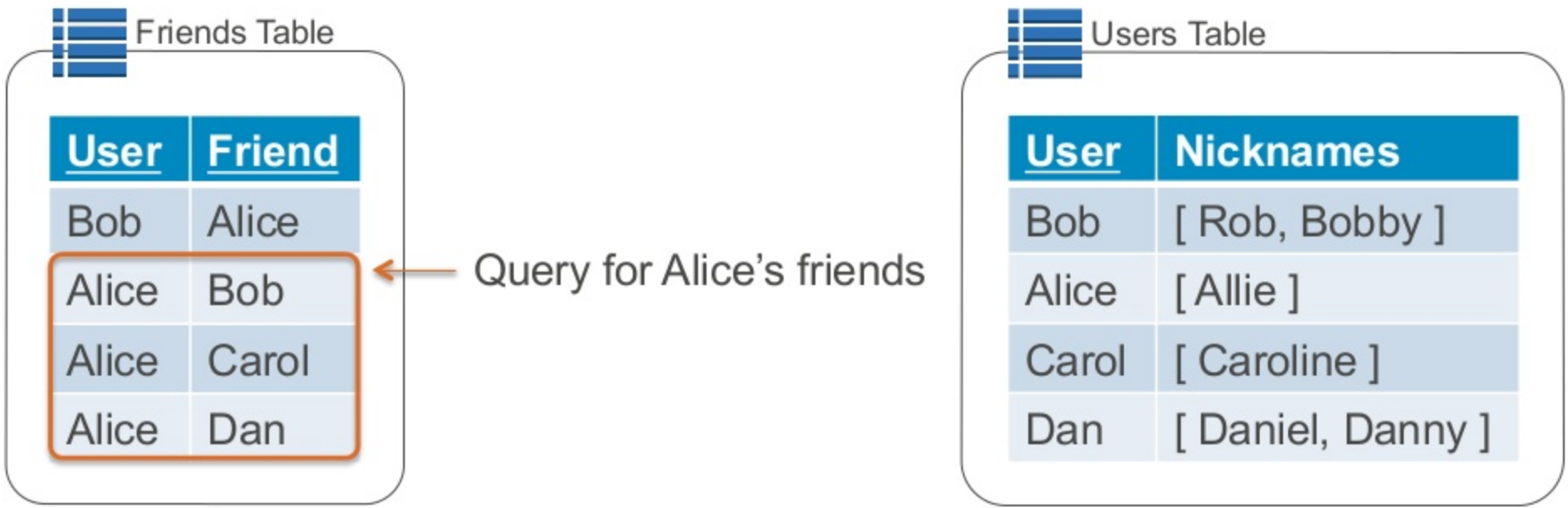
Users Table

User	Nicknames
Bob	[Rob, Bobby]
Alice	[Allie]
Carol	[Caroline]
Dan	[Daniel, Danny]

Social Network



Social Network



Recap: Social Network

- Hash schema for key-value lookups
- Hash and Range schema for Query

Image Tagging

secondary indexes

Image Tagging

- Query a user’s images
- Query a user’s images by date
- Tag other users in images
- Query images a user is tagged in

Image Tagging

Images Table

Image Tagging

Images Table

User	Image	Date	Link
Bob	aed4c	2013-10-01	s3://…
Bob	cf2e2	2013-09-05	s3://…
Bob	f93bae	2013-10-08	s3://…
Alice	ca61a	2013-09-12	s3://…

Image Tagging

Hash and Range
Primary Key Schema

Images Table

User	Image	Date	Link
Bob	aed4c	2013-10-01	s3://…
Bob	cf2e2	2013-09-05	s3://…
Bob	f93bae	2013-10-08	s3://…
Alice	ca61a	2013-09-12	s3://…

Image Tagging

Bob

Images Table

User	Image	Date	Link
Bob	aed4c	2013-10-01	s3://…
Bob	cf2e2	2013-09-05	s3://…
Bob	f93bae	2013-10-08	s3://…
Alice	ca61a	2013-09-12	s3://…

Query for Bob's Images

Image Tagging

- Query a user’s images
- **Query a user’s images by date**
- Tag other users in images
- Query images a user is tagged in

Local Secondary Indexes

- Alternate Range Key for your table
- More flexible Query patterns
- Local to the Hash Key

Image Tagging

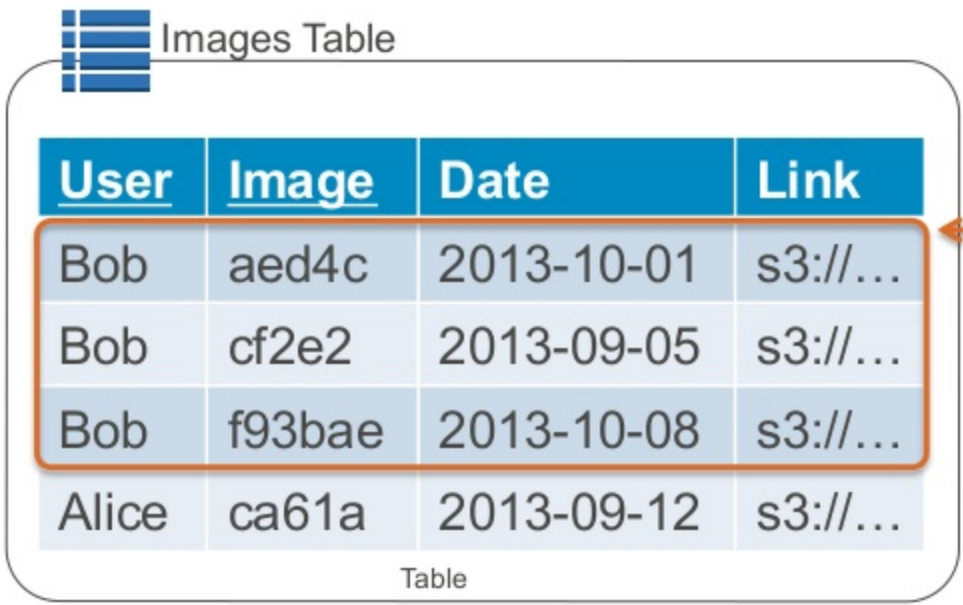


Image Tagging

Local Secondary Index on Date

Images Table

User	Image	Date	Link
Bob	aed4c	2013-10-01	s3://…
Bob	cf2e2	2013-09-05	s3://…
Bob	f93bae	2013-10-08	s3://…
Alice	ca61a	2013-09-12	s3://…

Table

User	Date	Image
Bob	2013-09-05	cf2e2
Bob	2013-10-01	aed4c
Bob	2013-10-08	f93bae
Alice	2013-09-12	ca61a

ByDate Local Secondary Index

Image Tagging

Query for Bob's two most recent images

Images Table

User	Image	Date	Link
Bob	aed4c	2013-10-01	s3://…
Bob	cf2e2	2013-09-05	s3://…
Bob	f93bae	2013-10-08	s3://…
Alice	ca61a	2013-09-12	s3://…

Table

User	Date	Image
Bob	2013-09-05	cf2e2
Bob	2013-10-01	aed4c
Bob	2013-10-08	f93bae
Alice	2013-09-12	ca61a

ByDate Local Secondary Index

Image Tagging

- Query a user’s images
- Query a user’s images by date
- **Tag other users in images**
- Query images a user is tagged in

Image Tagging

Image Tagging

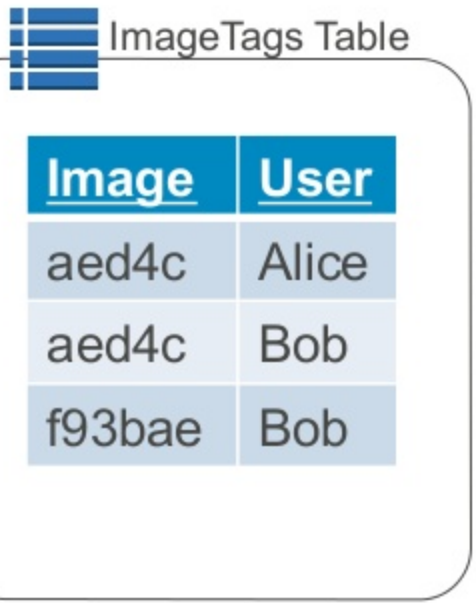


Image Tagging

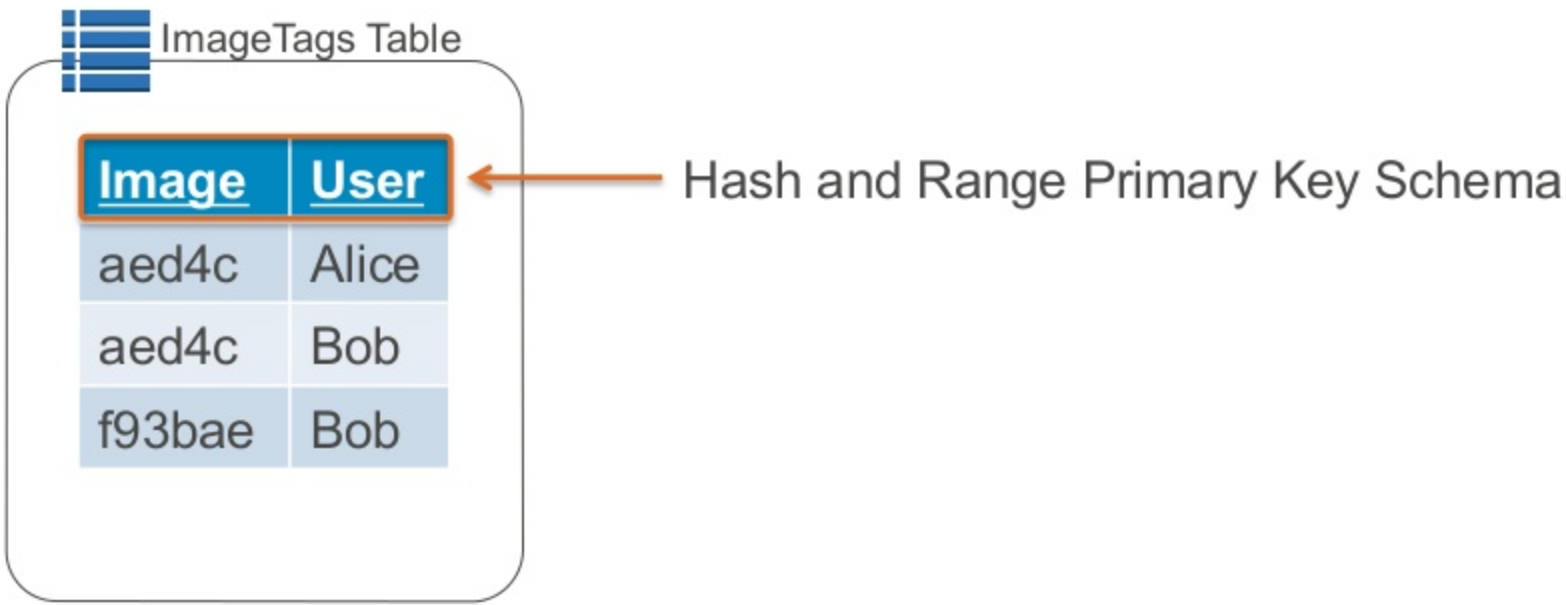


Image Tagging

Bob

ImageTags Table

Image	User
aed4c	Alice
aed4c	Bob
f93bae	Bob

Query Users
tagged in Image aed4c

Image Tagging

Bob

ImageTags Table

Image	User
aed4c	Alice
aed4c	Bob
f93bae	Alice
f93bae	Bob

Tag Alice in Image f93bae

Recap: Image Tagging

- Query a user's images
- Query a user's images by date
- Tag other users in images
- **Query images a user is tagged in**

Global Secondary Indexes

- Alternate Hash and/or Range Key for your table
- Even more flexible Query patterns

Image Tagging

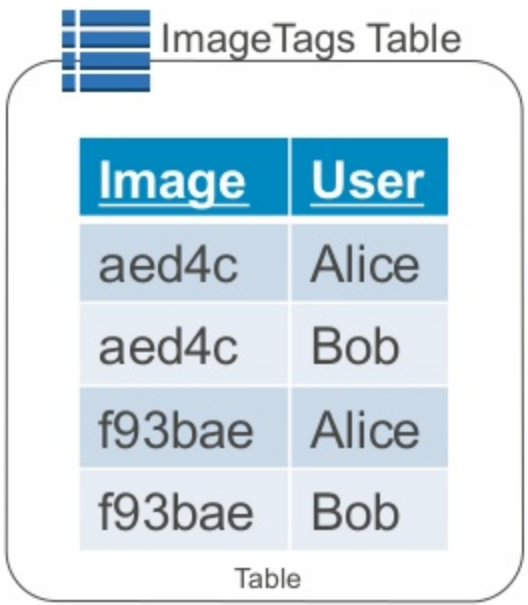


Image Tagging

Global Secondary Index on User, Image

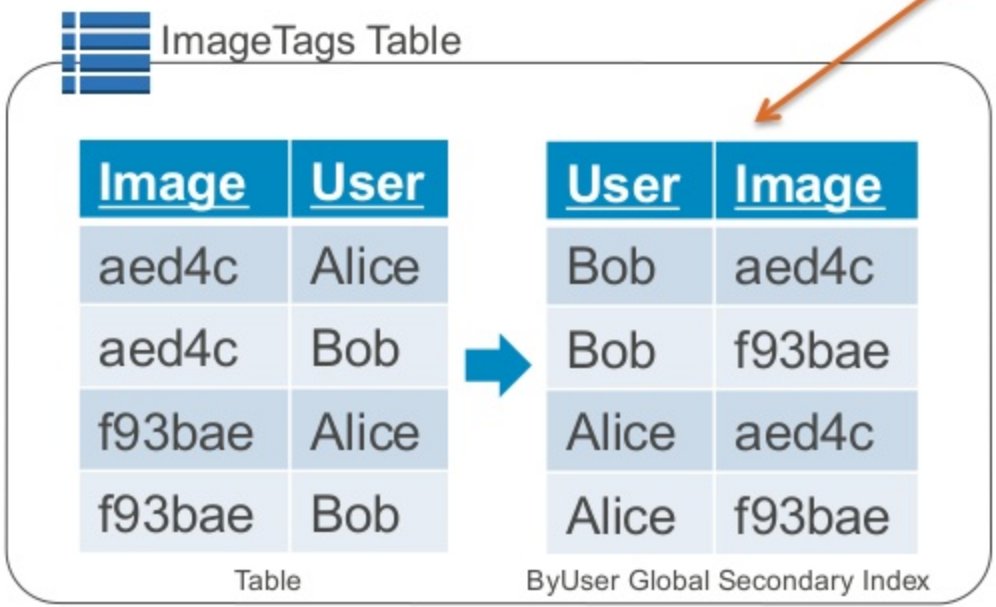


Image Tagging

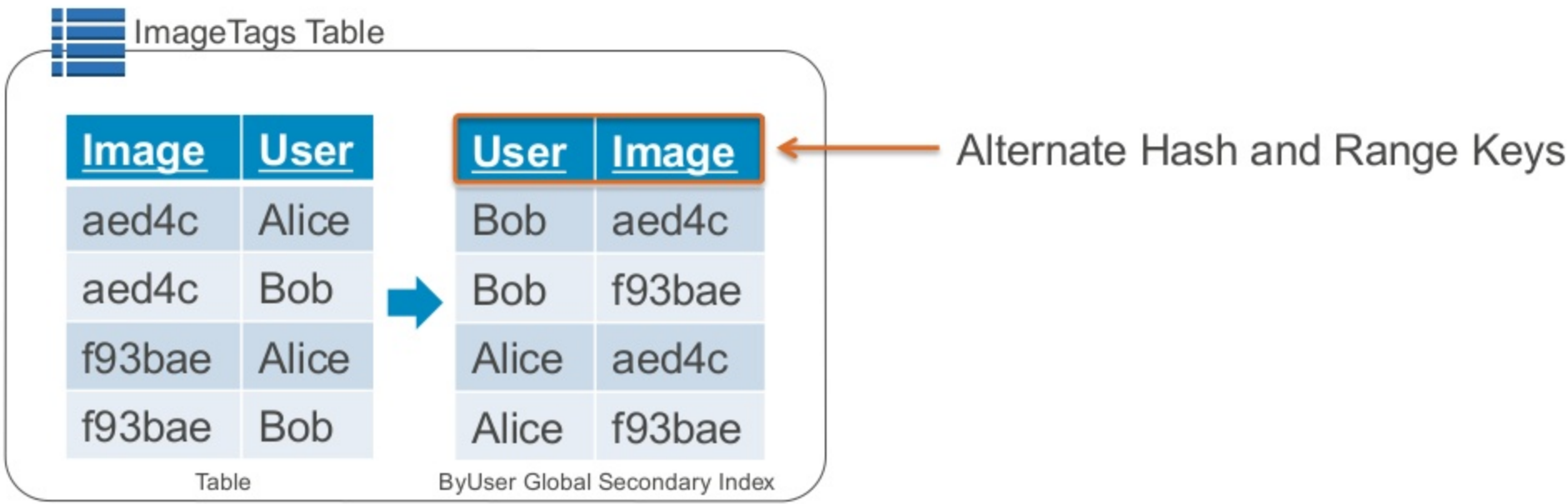
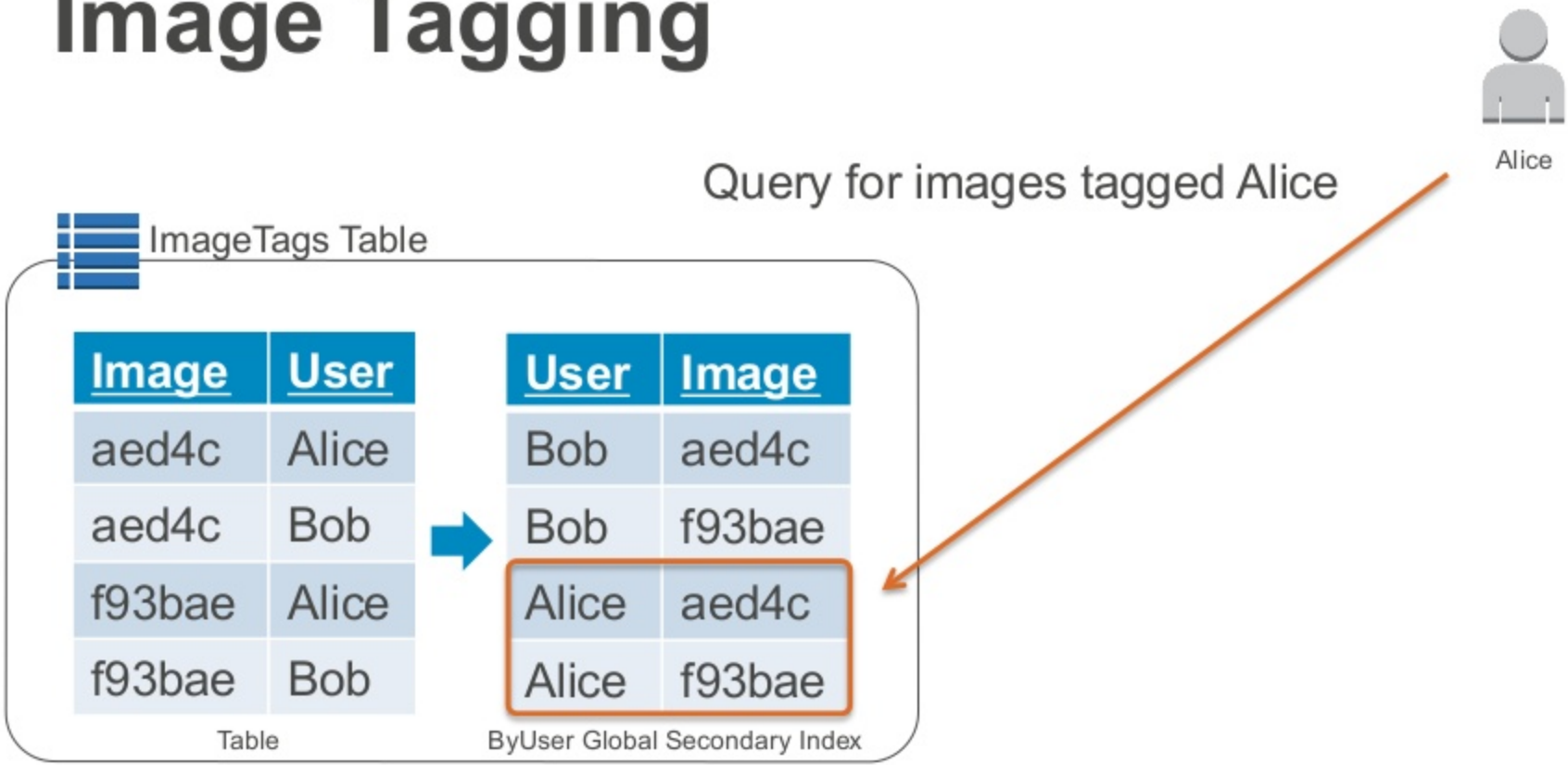


Image Tagging



Recap: Image Tagging

- Local Secondary Indexes support flexible queries
- Global Secondary Indexes unlock even more flexible queries

EXAMPLE 3:

Storing large items

Unlimited storage

Unlimited attributes per item.

Unlimited items per table.

Maximum of 64k per item.

Split across items.

message_id = 1	part = 1	message = <first 64k>
message_id = 1	part = 2	message = <second 64k>
message_id = 1	part = 3	joined = <third 64k>

Store a pointer to S3.

message_id = 1	message = http://s3.amazonaws.com...
message_id = 2	message = http://s3.amazonaws.com...
message_id = 3	message = http://s3.amazonaws.com...

EXAMPLE 4:

Time Series Data

Hot and cold tables.

April

event_id = 1000	timestamp = 2013-04-16-09-59-01	key = value
event_id = 1001	timestamp = 2013-04-16-09-59-02	key = value
event_id = 1002	timestamp = 2013-04-16-09-59-02	key = value

March

event_id = 1000	timestamp = 2013-03-01-09-59-01	key = value
event_id = 1001	timestamp = 2013-03-01-09-59-02	key = value
event_id = 1002	timestamp = 2013-03-01-09-59-02	key = value


December
January
February
March
April

Archive data

Move old data to S3: lower cost.
Still available for analytics.

Run queries across hot and cold data
with Elastic MapReduce.

Partitioning

Uniform workload

Data stored across multiple partitions.
Data is primarily distributed by primary key.

Provisioned throughput is divided evenly across partitions.

To achieve and maintain full provisioned throughput, spread workload evenly across hash keys

Non-Uniform workloads

Might be throttled, even at high levels of throughput

BEST PRACTICE 1:

Distinct values for hash keys

Hash key elements should have a
high number of distinct values

Lots of users with unique user_id.

Workload well distributed across hash key.

user_id = mza	first_name = Matt	last_name = Wood
user_id = jeffbarr	first_name = Jeff	last_name = Barr
user_id = werner	first_name = Werner	last_name = Vogels
user_id = simone	first_name = Simone	last_name = Brunozzi
…	…	…

BEST PRACTICE 2:

Avoid limited hash key values

Hash key elements should have a high number of distinct values.

Small number of status codes.

Unevenly, non-uniform workload.

status = 200	date = 2012-04-01-00-00-01
status = 404	date = 2012-04-01-00-00-01
status 404	date = 2012-04-01-00-00-01
status = 404	date = 2012-04-01-00-00-01

BEST PRACTICE 3:

Model for even distribution

Access by hash key value should be evenly distributed across the dataset.

Large number of devices.

Small number which are much more popular than others.

Workload unevenly distributed.

mobile_id = 100	access_date = 2012-04-01-00-00-01
mobile_id = 100	access_date = 2012-04-01-00-00-02
mobile_id = 100	access_date = 2012-04-01-00-00-03
mobile_id = 100	access_date = 2012-04-01-00-00-04
...	...

Sample access pattern.

Workload randomized by hash key.

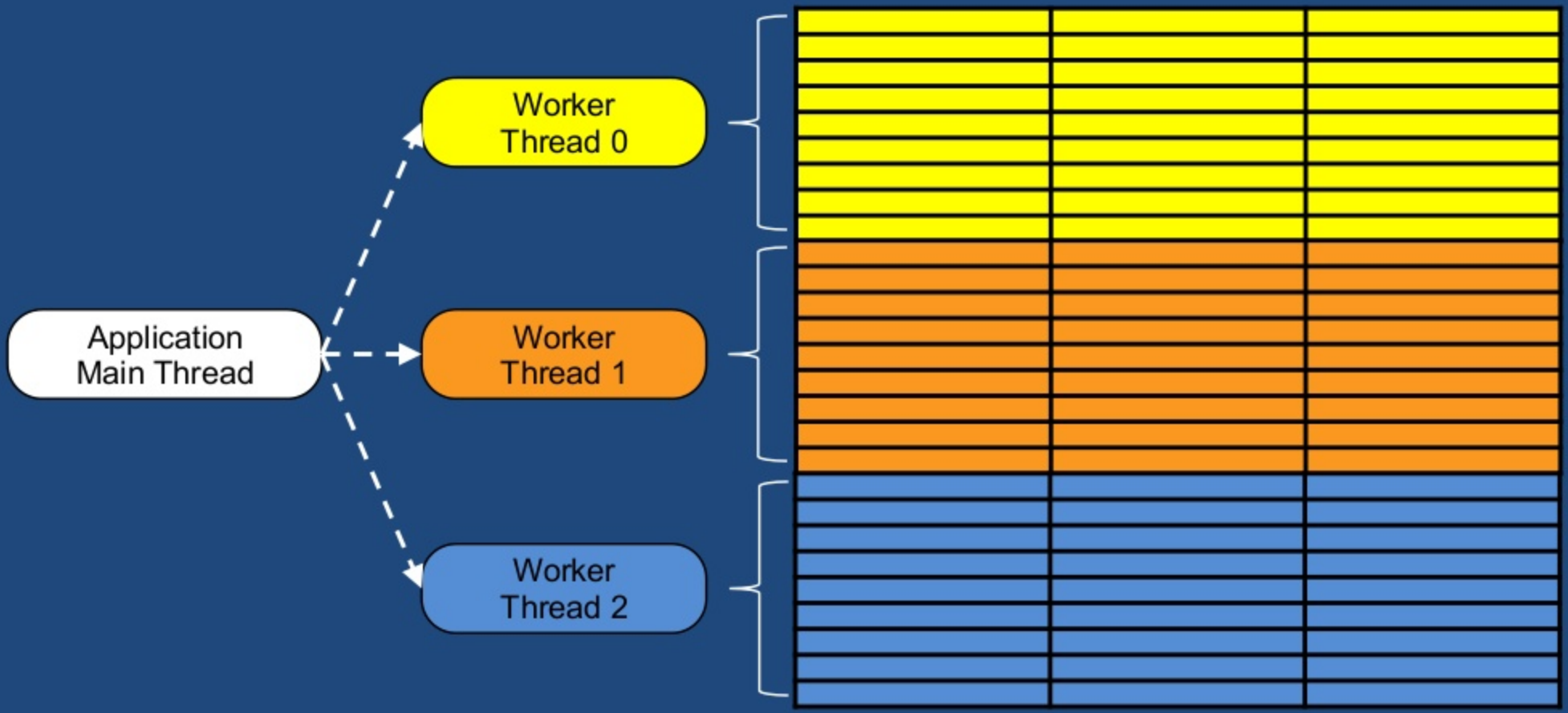
mobile_id = 100.1	access_date = 2012-04-01-00-00-01
mobile_id = 100.2	access_date = 2012-04-01-00-00-02
mobile_id = 100.3	access_date = 2012-04-01-00-00-03
mobile_id = 100.4	access_date = 2012-04-01-00-00-04
...	...

BEST PRACTICE 4:

Distribute scans across partitions

Improve retrieval times by scanning partitions concurrently using the Parallel Scan feature.

Parallel Scan: separate thread for each table segment



Working with the AWS CLI & SDKs

Developer Resources: http://aws.amazon.com/dynamodb/developer-resources/

AWS CLI for DynamoDB

Support for ad hoc operations, such as creating or deleting tables.

You can also use the CLI to embed DynamoDB operations within utility scripts

Working with the AWS CLI

```
aws dynamodb create-table
     --table-name MusicCollection
     --attribute-definitions
          AttributeName=Artist,AttributeType=S AttributeName=SongTitle,AttributeType=S
     --key-schema AttributeName=Artist,KeyType=HASH AttributeName=SongTitle,KeyType=RANGE
     --provisioned-throughput ReadCapacityUnits=1,WriteCapacityUnits=1
```

This example creates a DynamoDB table named MusicCollection with two string attributes per item and these attributes set as a hash key and a range key. Provisioned throughput is set at 1 read capacity unit & 1 write capacity unit.

For readability, the command is broken into several lines.

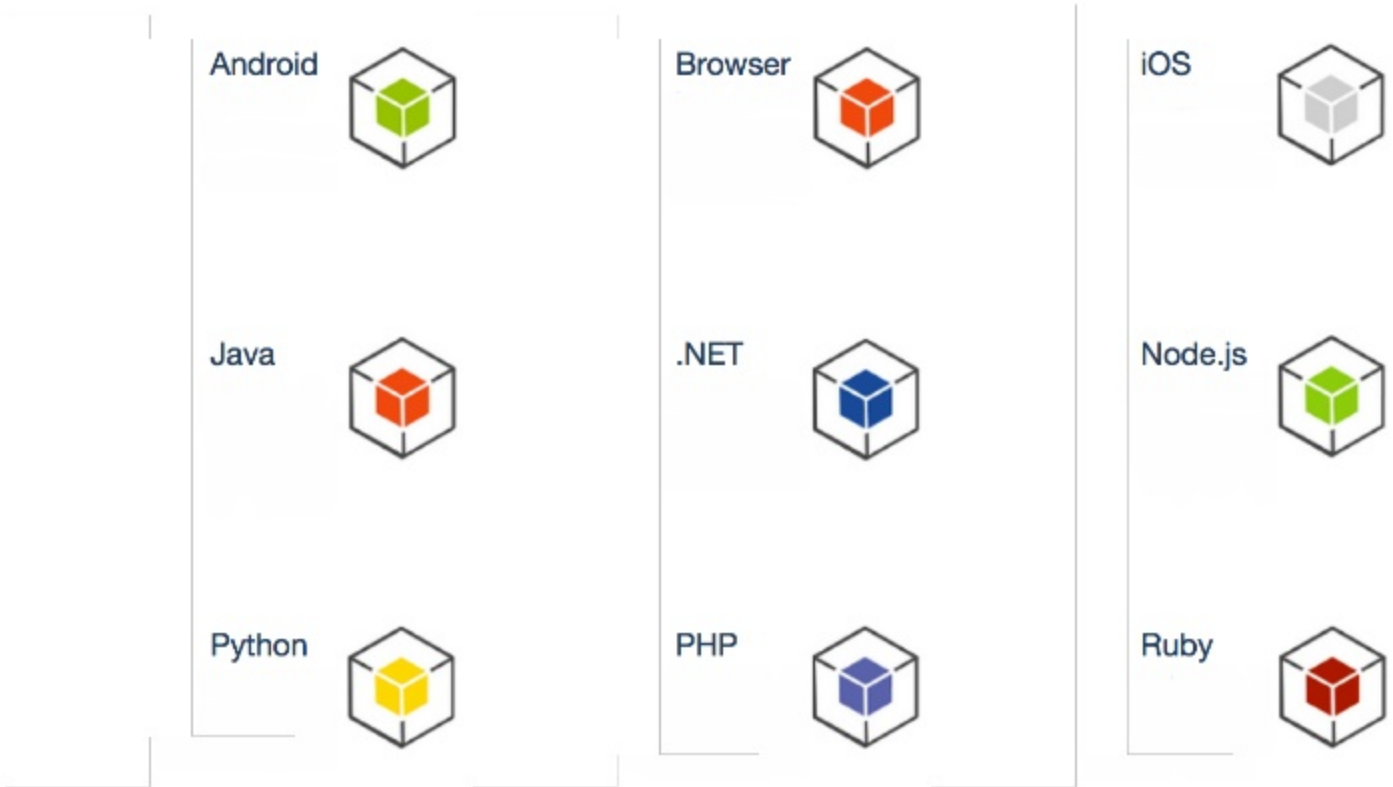
AWS CLI actions for DynamoDB

```
batch-get-item      delete-table       put-item           update-table
batch-write-item    describe-table     query
create-table        get-item           scan
delete-item         list-tables        update-item
```

As usual, you can get more details via the AWS CLI

```
$ aws dynamodb get-item help
```

Working with the AWS SDKs



Simplify using Amazon DynamoDB in your applications with an API tailored to your programming language or platform
Comprehensive documentation for all these SDKs is available here: aws.amazon.com/dynamodb/developer-resources/

Mobile SDKs

Amazon DynamoDB Object Mapper

Map your client-side classes to Amazon DynamoDB tables without having to write the code to transform objects into tables and vice versa

More detail on the mobile SDK is available here: aws.amazon.com/mobile/sdk/

Working with DynamoDB Local

A small client-side database and server that mimics the DynamoDB service.

Enables you to write applications that use the DynamoDB API, without actually manipulating any tables or data in DynamoDB.

All API actions are rerouted to DynamoDB Local. When your application creates a table or modifies data, those changes are written to a local database. This lets you save on provisioned throughput, data storage, and data transfer fees whilst developing your applications.

More details: aws.amazon.com/amazondynamodb/latest/developerguide/Tools.DynamoDBLocal.html

3rd Party Tools, Libraries, Mappers & Mocks for DynamoDB

Autoscaling: Dynamic DynamoDB (AWS Blog Post)

Testing: 3rd party emulators/mocks for development and testing

Backup/Archiving: DynamoDB Table Archiver

3rd Party Language Libraries: including – Coldfusion, Django, Erlang, Go, Groovy/Grails, Java, JavaScript, .NET, Node.js, Perl, Python & Ruby

3rd party tools are listed on the Developer Resources page: http://aws.amazon.com/dynamodb/developer-resources/

Reporting & Analysis

Seamless scale

Scalable methods for data processing.

Scalable methods for backup/restore.

Amazon Elastic MapReduce

Managed Hadoop service for data-intensive workflows

aws.amazon.com/emr

Integrating data from DynamoDB with Amazon EMR + Hive

```
create external table items_db
  (id string, votes bigint, views bigint) stored by
  'org.apache.hadoop.hive.dynamodb.DynamoDBStorageHandler'
   tblproperties
  ("dynamodb.table.name" = "items",
   "dynamodb.column.mapping" =
   "id:id,votes:votes,views:views");
```

This is the same process as mapping an external table from data stored in HDFS, but it uses the AWS provided storage handler named dynamodb.DynamoDBStorageHandler

Integrating data from DynamoDB with Amazon EMR + Hive

```
select id, likes, views
from items_db
order by views desc;
```

This is the same process as mapping an external table from data stored in HDFS, but it uses the AWS provided storage handler named dynamodb.DynamoDBStorageHandler

Some other things…

Who is using DynamoDB?

> We spend more on snacks than we do on Amazon DynamoDB.
>
> **Valentino Volonghi**
> CTO, Adroll

AdRoll, an online advertising platform, serves 50 billion impressions a day worldwide with its global retargeting platforms.

Adroll Uses AWS to Grow by More Than 15,000% in a Year

- Needed high-performance, flexible platform to swiftly sync data for worldwide audience
- Processes 50 TB of data a day
- Serves 50 billion impressions a day
- Stores 1.5 PB of data
- Worldwide deployment minimizes latency

Find out more about this and other AWS case studies here: aws.amazon.com/solutions/case-studies/

Resources to learn more

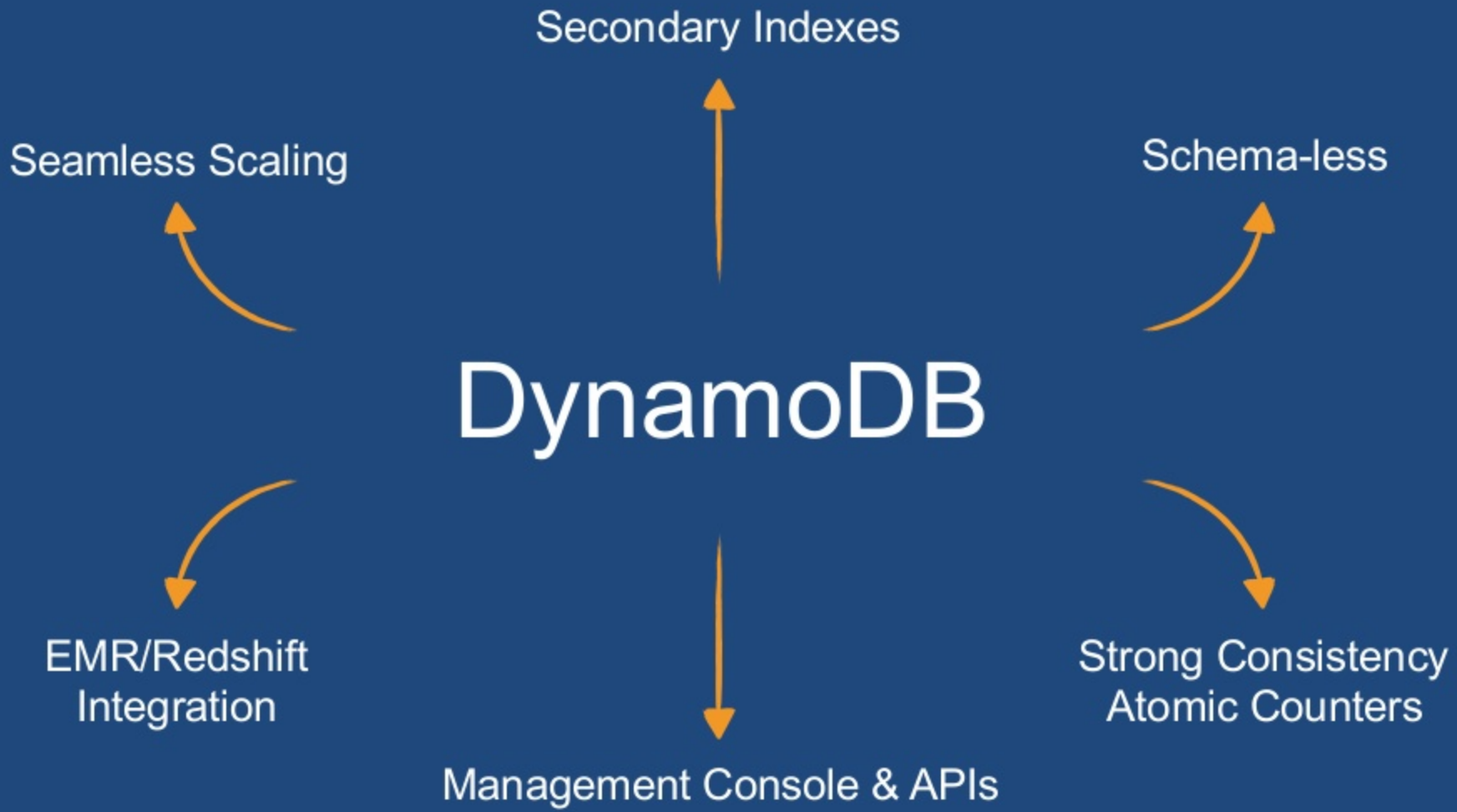
Getting Started with Amazon DynamoDB (AWS Documentation Center):
docs.aws.amazon.com/amazondynamodb/latest/developerguide/GettingStartedDynamoDB.html

Auto Scale DynamoDB with Dynamic DynamoDB (Jeff Barr's Blog):
aws.amazon.com/blogs/aws/auto-scale-dynamodb-with-dynamic-dynamodb/

SmugMug: From MySQL to Amazon DynamoDB (DAT204) from AWS re:Invent 2013 (video)
https://www.youtube.com/watch?v=g1Yk0E6sslg

Amazon DynamoDB Design Patterns for Ultra-High Performance Apps (DAT304) from AWS re:Invent 2013 (video)
https://www.youtube.com/watch?v=Dh8kp1AcRg0

Summary


Secondary Indexes
Seamless Scaling
Schema-less
DynamoDB
EMR/Redshift
Integration
Strong Consistency
Atomic Counters
Management Console & APIs

Find out more:

aws.amazon.com/dynamodb

AWS Training & Certification

Self-Paced Labs

Try products, gain new skills, and get hands-on practice working with AWS technologies

aws.amazon.com/training/self-paced-labs

Training

Skill up and gain confidence to design, develop, deploy and manage your applications on AWS

aws.amazon.com/training

Certification

Demonstrate your skills, knowledge, and expertise with the AWS platform

aws.amazon.com/certification

amazon web services

Follow us for more events & webinars

Ian Massingham – Technical Evangelist
@IanMmmm

@AWS_UKI for local AWS events & news

@AWScloud for Global AWS News and Announcements

We typically see customers start by trying our services

All Products

Compute & Networking

Storage

Database

Application Services

Development & Management

AWS Marketplace Software

FAQ »

Find answers to common questions about the AWS Free Tier.

Amazon EC2 »
Web service that provides resizable compute capacity in the cloud.

Amazon S3 »
Highly-scalable, reliable, and low-latency data storage.

Amazon RDS »
Managed MySQL, Oracle and SQL Server databases.

Amazon CloudWatch »
Monitoring for AWS cloud resources and applications.

AWS Data Pipeline »
Orchestration for data-driven workflows.

Amazon DynamoDB »
Fully managed NoSQL database service with seamless scalability.

Amazon EBS »
Highly available, highly reliable, predictable storage volumes.

Amazon ELB »
Web service that provides scalability and high availability.

Amazon ElastiCache »
Managed scale-out caching.

Amazon SNS »
Web service to set up, operate, and send notifications from the cloud.

Amazon Elastic Transcoder »
Convert your media files easily, at low cost and at scale.

Amazon SQS »
Scalable queue for storing messages as they travel between computers.

Amazon SWF »
Workflow service for building scalable, resilient applications.

AWS Marketplace »
Partner software pre-configured to run on AWS.

Get started now at : aws.amazon.com/getting-started

amazon web services

Design your application for the AWS Cloud

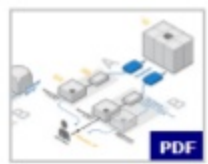
AWS Reference Architectures

The flexibility of AWS allows you to design your application architectures the way you like. AWS Reference Architecture Datasheets provide you with the architectural guidance you need in order to build an application that takes full advantage of the AWS cloud. Each datasheet includes a visual representation of the architecture and basic description of how each service is used.

Web Application Hosting
Build highly-scalable and reliable web or mobile-web applications (PDF)

Content and Media Serving
Build highly reliable systems that serve massive amounts of content and media (PDF)

Batch Processing
Build auto-scalable batch processing systems like video processing pipelines (PDF)



Fault tolerance and High Availability
Build systems that quickly failover to new instances in an event of failure (PDF)

Large Scale Processing and Huge Data sets
Build high-performance computing systems that involve Big Data (PDF)

Ad Serving
Build highly-scalable online ad serving solutions (PDF)

Disaster Recovery for Local Applications
Build cost-effective Disaster Recovery solutions for on-premises applications (PDF)

File Synchronization
Build simple file synchronization service (PDF)

More details on the AWS Architecture Center at : aws.amazon.com/architecture

amazon web services