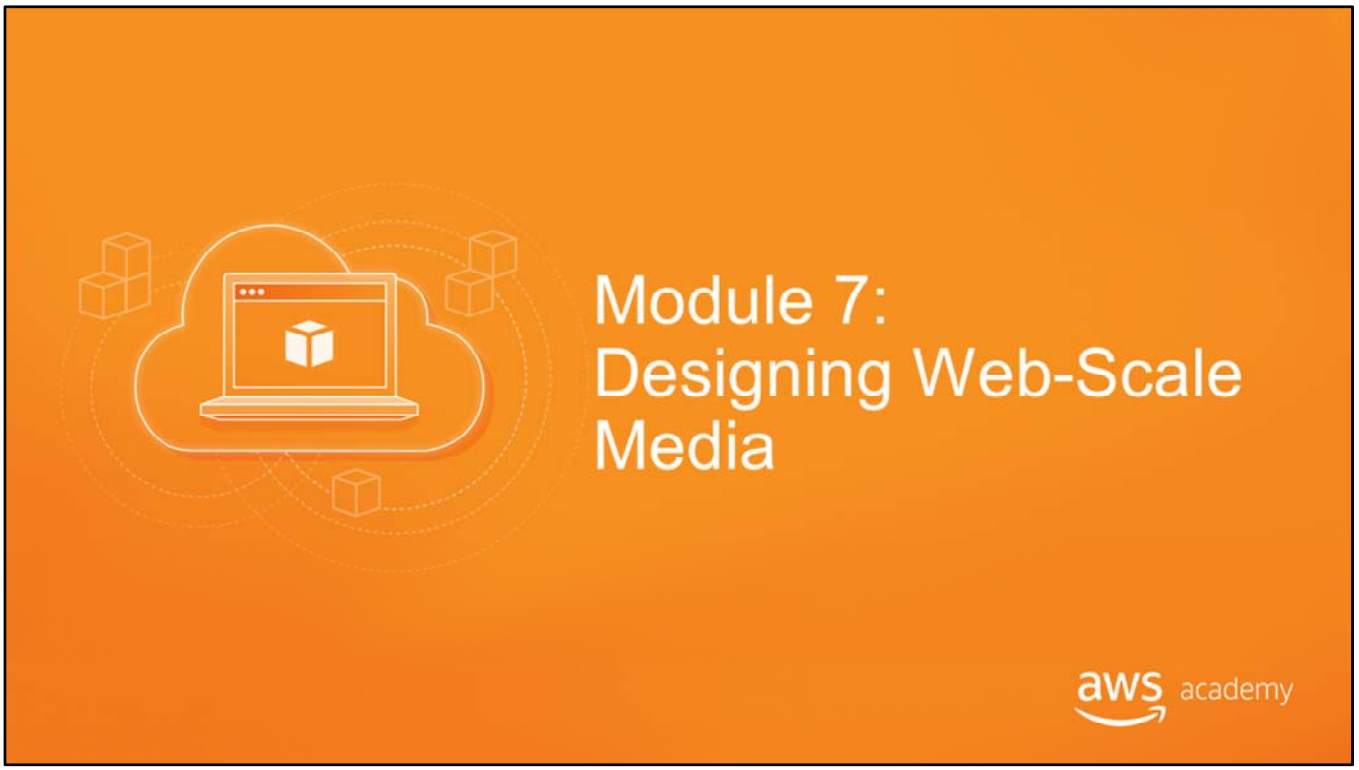


Welcome to Module 7: Designing Web-Scale Media.

What's in This Module

- **Part 1:** Storing Web-Accessible Content with Amazon S3 (Week 3)
- **Part 2:** Caching with Amazon CloudFront (Week 10)
- **Part 3:** Managing NoSQL databases
- **Part 4:** Storing Relational Data in Amazon RDS.

In part 1, you will discover how to store web accessible content with Amazon Simple Storage Service, or Amazon S3.
In part 2, you will review caching with Amazon CloudFront.
In part 3, you will learn about the features of managing NoSQL databases, or DBs.
In part 4, we'll learn how to store relational data in Amazon Relational Database Service, or Amazon RDS.

You will finish this module by completing a lab: Implementing a Serverless Architecture with AWS Managed Services.

Introducing Part 1: Storing Web-Accessible Content with Amazon S3.

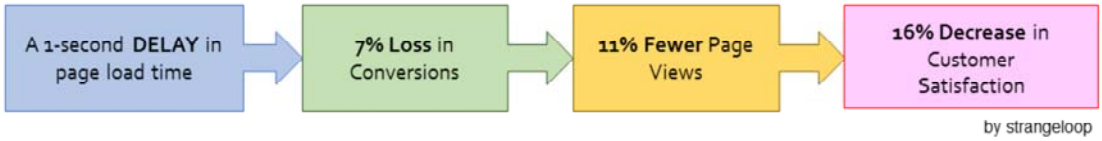
Reality of Web-Based Applications

Your applications being unavailable:

- Leads to revenue loss.
- Has an impact on customer loyalty and your brand image.

Performance translates to:

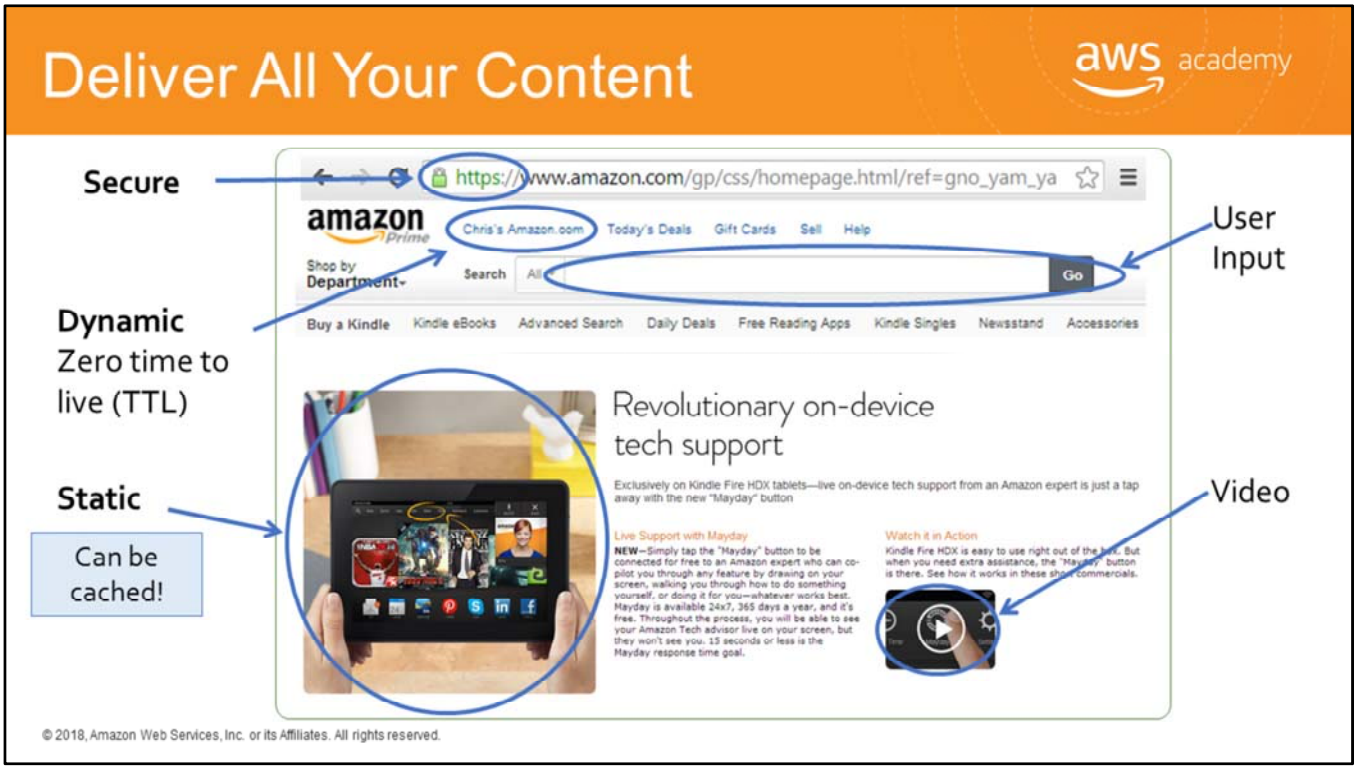
- High page views.
- Better customer experience.
- Higher conversion rates.



by strangeloop

Web-based applications must be available, and they shouldn't take long to load. If your applications are unavailable, it could lead to revenue loss. It could also affect customer loyalty and your brand image. A 1-second delay can translate into a 16 percent decrease in customer satisfaction.

Performance translates to higher page views, better customer experience, and higher conversion rates.



Let's take a moment to review Amazon.com and consider where we might deliver content.

In the address bar, you can note that the application is transported over https://, or port 443. This tells you that Amazon.com uses Secure Sockets Layer, or SSL. You can then note that the application content is dynamic because it's personalized for Chris, which tells you that it has zero time to live, or TTL. You can also search for items by entering user input. The static images can be cached and stored in Amazon S3.

With the caching and acceleration technology of Amazon CloudFront, we can deliver all of your content, including static images, videos, and content based on user input.

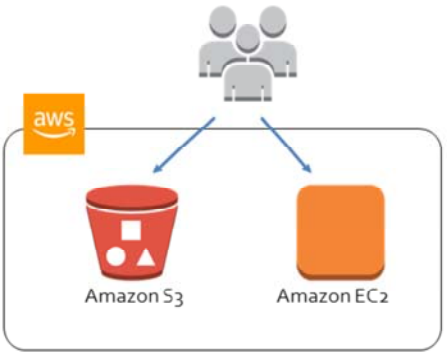
How should web-accessible content be stored? Let's discuss how to store content in Amazon S3.

Web Storage Pattern

Static assets—which are assets that aren't going to change, such as text, images, and video content—can be stored in Amazon S3.

Problem

- Delivery of large files from a web server can become a problem in terms of network latency.
- User-generated content needs to be distributed across all your web servers.



Static assets—content that isn't going change, such as text, images, and video content–can be stored in Amazon S3.

Let's consider this pattern. An Amazon Elastic Compute Cloud—or Amazon EC2—instance is being used to serve static images and videos, which is not a proper use for that Amazon EC2 instance. Instead, you can store those images and videos in an Amazon S3 bucket that is enabled for public web hosting. You know that Amazon S3 will be durable, and that it will serve content quickly. When you use Amazon S3 to store static content, you don't need to worry about scaling out an Amazon EC2 instance to serve videos and images.

Delivery of large files from a web server can become a problem in terms of network latency.

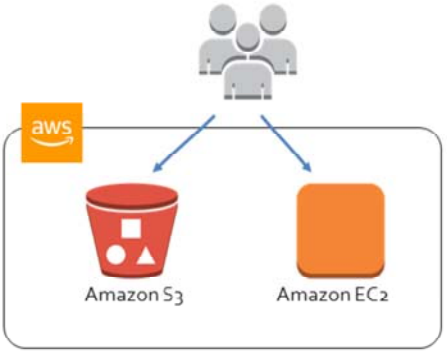
The use of Amazon S3 eliminates the need to worry about network loads and data capacity on your web servers. You can also use Amazon CloudFront, which is a content delivery network—or CDN—that can be used as a global caching layer in front of Amazon S3 to accelerate content to your end users. User-generated content needs to be distributed across all your web servers.

Web Storage Pattern

aws academy

Solution

- Store static asset files (CSS, multimedia, etc.) in Amazon S3 and deliver the files from there.
- Objects that are stored in Amazon S3 can be accessed directly by users if set to public.
- Amazon S3 performs backups in at least three different data centers for extremely high durability.



One solution is to store static asset files—such as Cascading Style Sheets, or CSS; multimedia, and others—in Amazon S3 and serve the files from there.

Objects that are stored in Amazon S3 can be accessed directly by users if the bucket is public.

Amazon S3 performs backups in at least three different data centers, and thus has extremely high durability.

Choosing a Region

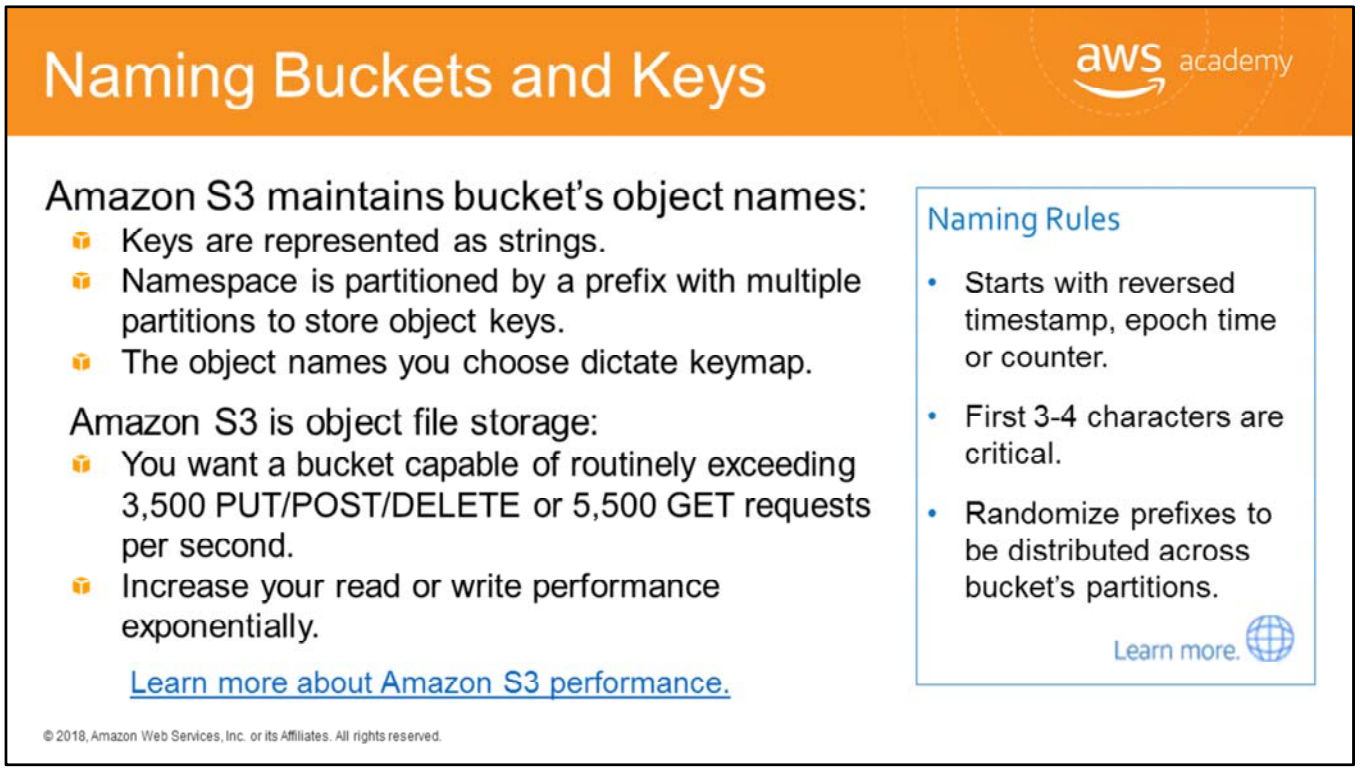
aws academy

Choosing a Region:

- Performance depends on the proximity to your users.
- Collocating with compute and other AWS resources affects performance.

How do you choose a Region for Amazon S3? You will want to have geographical proximity to your users. You should also collocate it with your Amazon EC2 instances, and with other AWS services that upload and download bucket objects. It is better to locate services as close as possible to the Amazon S3 buckets that must accessed. Collocating all of your relevant AWS services and Amazon S3 buckets in the same Region can help reduce costs and latencies.

Data that is stored in an Amazon S3 bucket is replicated across multiple data centers in a geographical region. It is important to carefully select the AWS Region where you store an object because response latencies grow when requests have to travel long distances over the internet.



Amazon S3 maintains a "map" of each bucket's object names, or "keys." Internally, keys are all represented in Amazon S3 as strings with the format of bucket name/key name. The namespace is partitioned by a prefix. Each bucket contains multiple partitions to store object keys. Keys partition the bucket key name space, and define the partition that they are stored in.

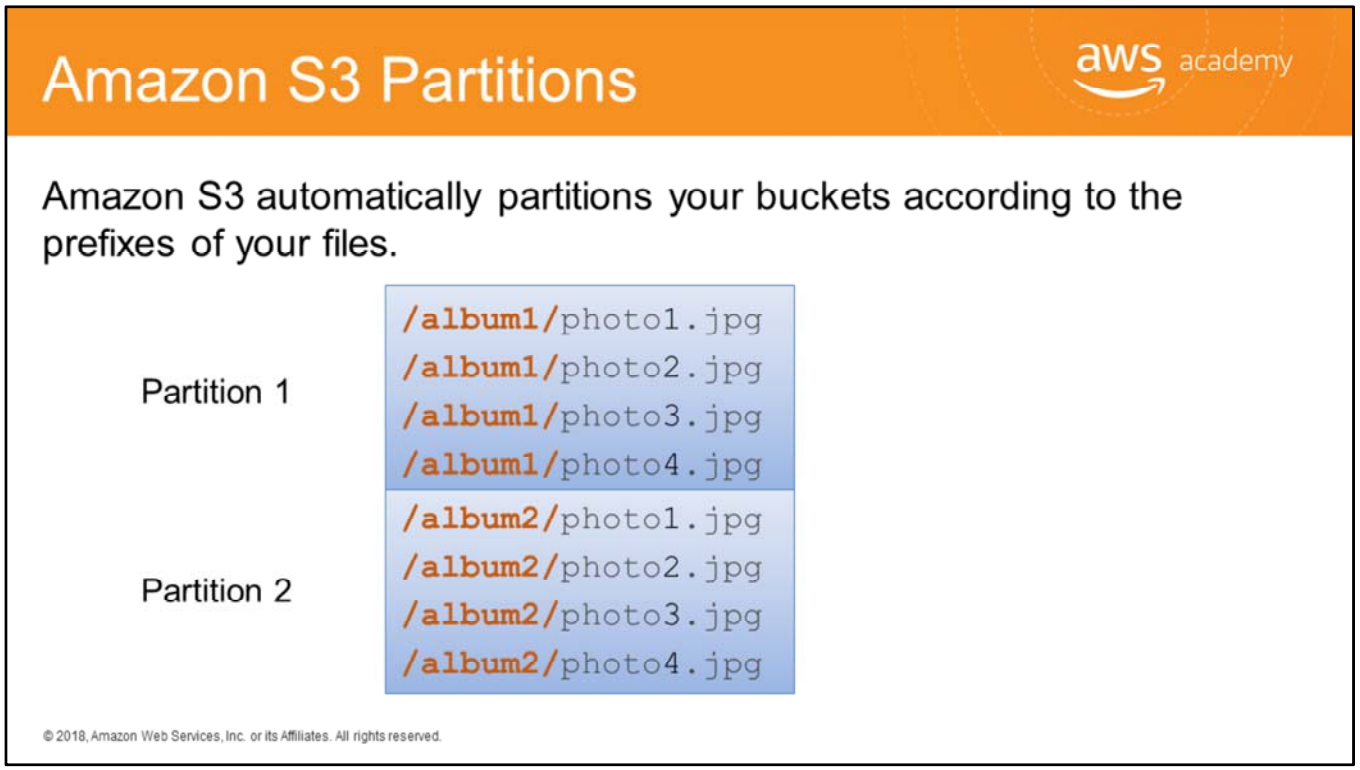
Each key is stored and retrieved based on the name that is provided when the object is first put into Amazon S3, which means that the object names that you choose actually dictate the keymap. Naming is critical when you want to achieve a processing rate of more than 100 transactions per second—or TPS—for the Amazon S3 bucket. Although buckets can be named with any alphanumeric character, you should follow some naming rules to ensure that you can reference your bucket by using the following convention: <bucketname>.s3.amazonaws.com. What makes a good or a bad Amazon S3 key? Plain globally unique identifiers, or GUIDs; timestamps; and running counters are not good candidates for key names because they use repeating prefixes. A good key starts with a reversed timestamp, a reversed epoch time, or a reversed counter. The most significant part of the name is the first 3-4 characters of a key. The best practice for creating good Amazon S3 keys is to randomize prefixes as much as possible so they can be distributed across a bucket's partitions.

For additional information, go to the following page about bucket restrictions and naming: https://docs.aws.amazon.com/AmazonS3/latest/dev/BucketRestrictions.html

Amazon S3 is object file storage, and it behaves like a flat file system. It automatically scales to high request rates. For example, an application can achieve at least 3,500 PUT, POST, or DELETE

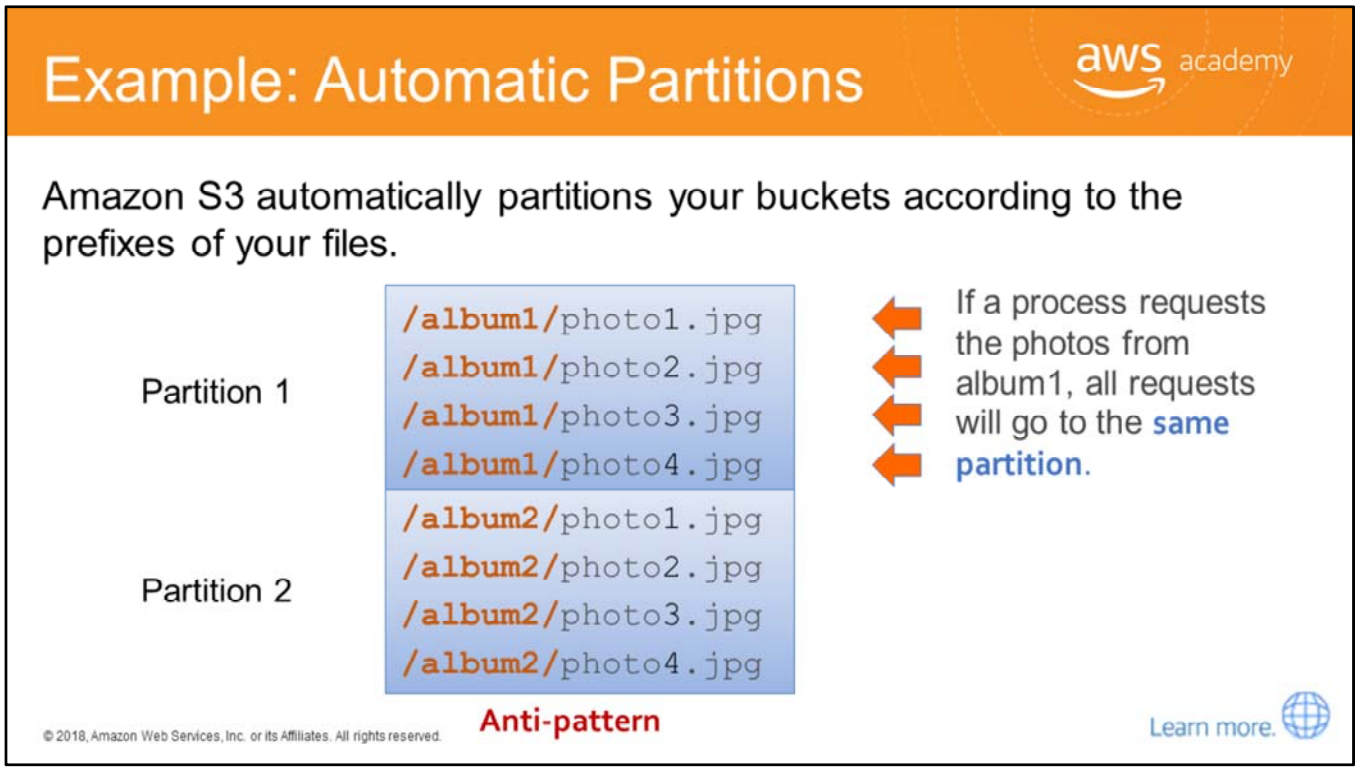
requests and 5,500 GET requests per second per prefix in a bucket. There are no limits to the number of prefixes in a bucket. It is simple to increase your read or write performance exponentially. For example, if you create 10 prefixes in an Amazon S3 bucket to parallelize reads, you could scale your read performance to 55,000 read requests per second.

For more information, go to the following page about Amazon S3 performance:
https://docs.aws.amazon.com/AmazonS3/latest/dev/request-rate-perf-considerations.html



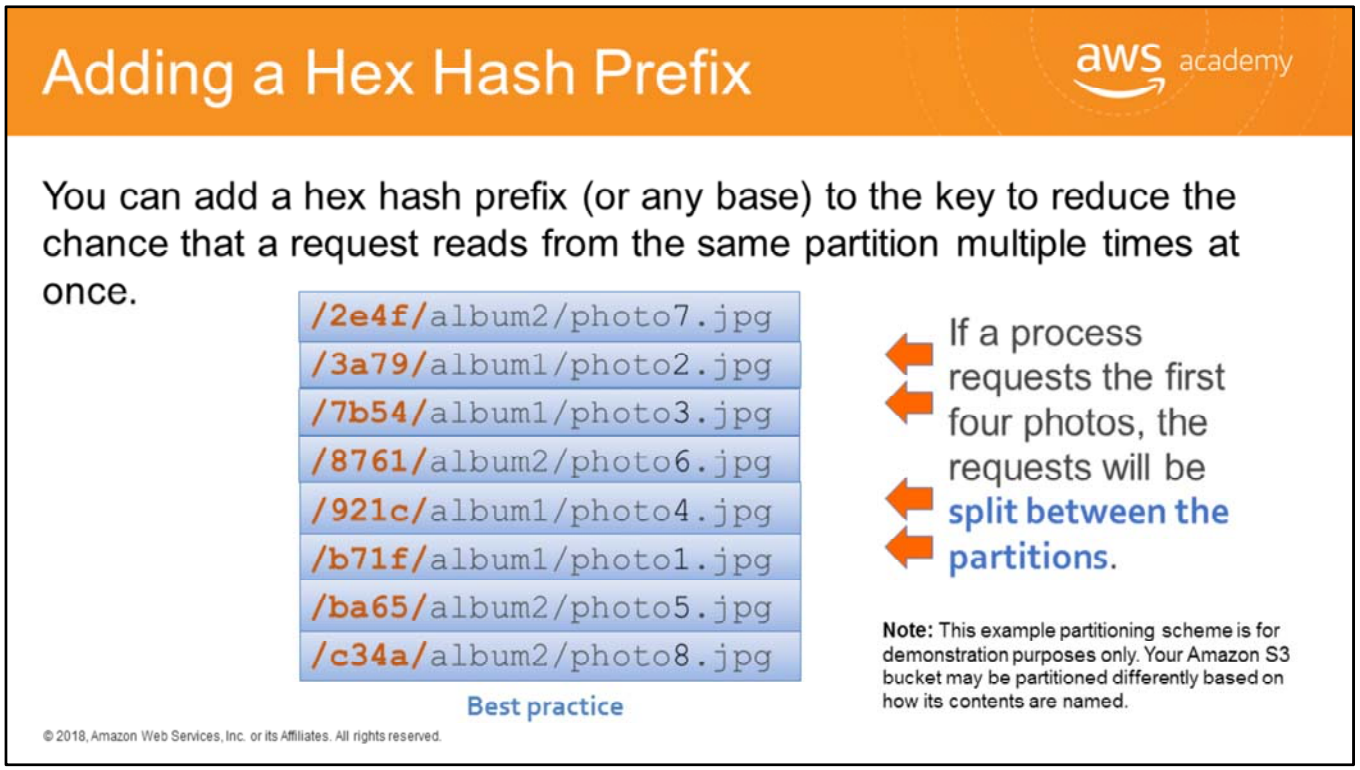
As mentioned previously, each bucket contains multiple partitions to store object keys. Amazon S3 automatically partitions your buckets according to the prefixes of your files. Keys partition the bucket keyname space, and they define the partition they are stored in. Additionally, keys in Amazon S3 are partitioned by prefix.

This example shows two partitions. The partitions are album1 and album2, which have photos. If you start having more than 300 requests per second, you can create a hotspot because many of the albums are named in the same way. They are all in the same partition because they are all in the same folder.



If you store a few objects and you need to add more objects, incrementing the last number of your file name is fine. If you store thousands of objects in an Amazon S3 bucket, this file naming convention could limit performance when Amazon S3 partitions your bucket. Adding a random value to the key can provide higher performance.

If you issue queries against Amazon S3 buckets that store a large number of objects and the data is not partitioned, these queries might affect the GET request rate limits in Amazon S3, and then lead to Amazon S3 exceptions. To prevent errors, you should partition your data. Additionally, consider tuning your Amazon S3 request rates. For more information, see Request Rate and Performance Considerations.



In the example of photo storage, if the Amazon S3 bucket receives more than 300 requests per second, the read request can have limited performance because it must increment numerically. For example, a query must begin at photo1.jpg when it needs to locate photo200.jpg.

Like the photo example, Amazon S3 workloads often have content that is organized by user ID, or game ID, album ID, or other similar semi-meaningless identifier. These identifiers often involve incrementally increasing numbers, or date-time constructs of various types. The difficulty of this naming choice for Amazon S3 scaling is twofold:

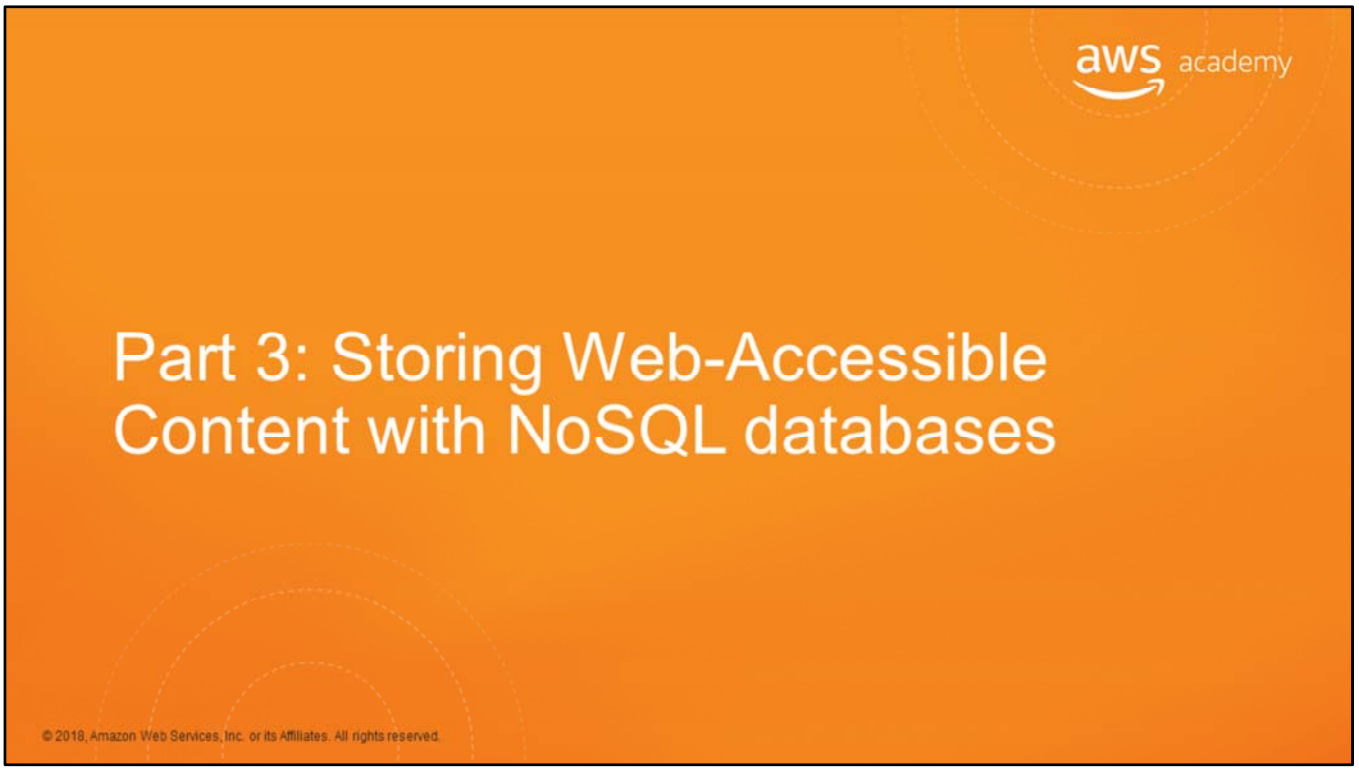
- First, all new content will necessarily end up being owned by a single partition.
- Second, all the partitions that hold slightly older—and generally less hot—content get cold more quickly than partitions that use other naming conventions. This practice effectively wastes the available operations per second that each partition can support by making all of the old content cold over time.

A way to make this work with Amazon S3 at nearly any request rate is to simply reverse the order of the digits in this identifier. In other words, you can use seconds of precision for date- or time-based identifiers. These identifiers effectively start with a random number, which fans out the transactions across many potential child partitions. Each of those child partitions scales, so that no meaningful operations per second budget are wasted. In fact, Amazon S3 even has an algorithm to detect this parallel type of write

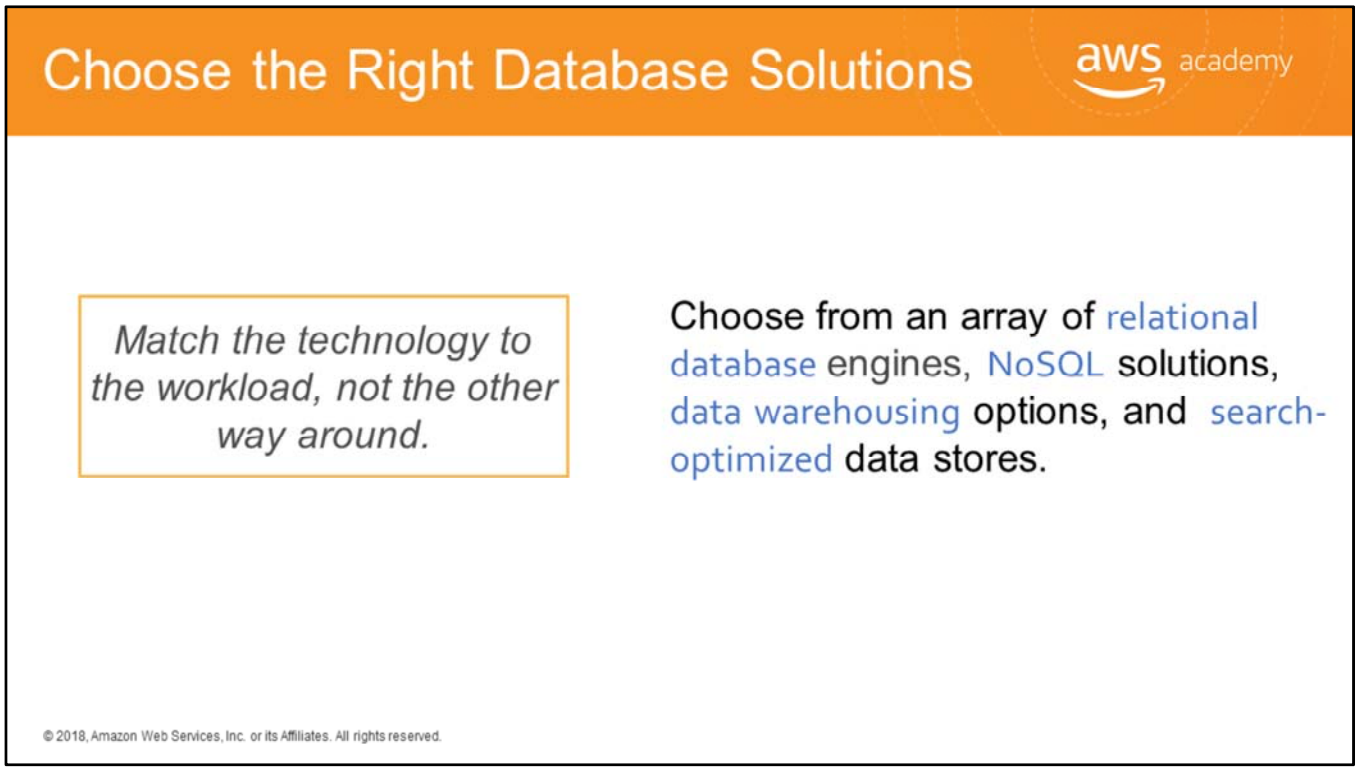
pattern, and it will automatically create multiple child partitions from the same parent simultaneously. This process increases the system's operations per second budget when request heat is detected.

Amazon S3 buckets are somewhat similar to Hashtable data structures in Java. Each bucket has multiple partitions to store objects keys, which helps spread the transaction load. You can add a hex hash prefix or any base in front of the bucket name to reduce the chance that a request reads from the same partition multiple times at once. If a process requests the first four photos, the requests will be split between the partitions. Larger bases provide better distribution.

In summary, keys partition the key name space in a bucket, and they define the partition that they are stored in. The most significant part is the first 3-4 characters of a key, and this number might increase along with the amount of objects that are stored in a bucket. The best practice of coming up with good Amazon S3 keys is to randomize their prefixes as much as possible so that they can be better distributed across a bucket's partitions.



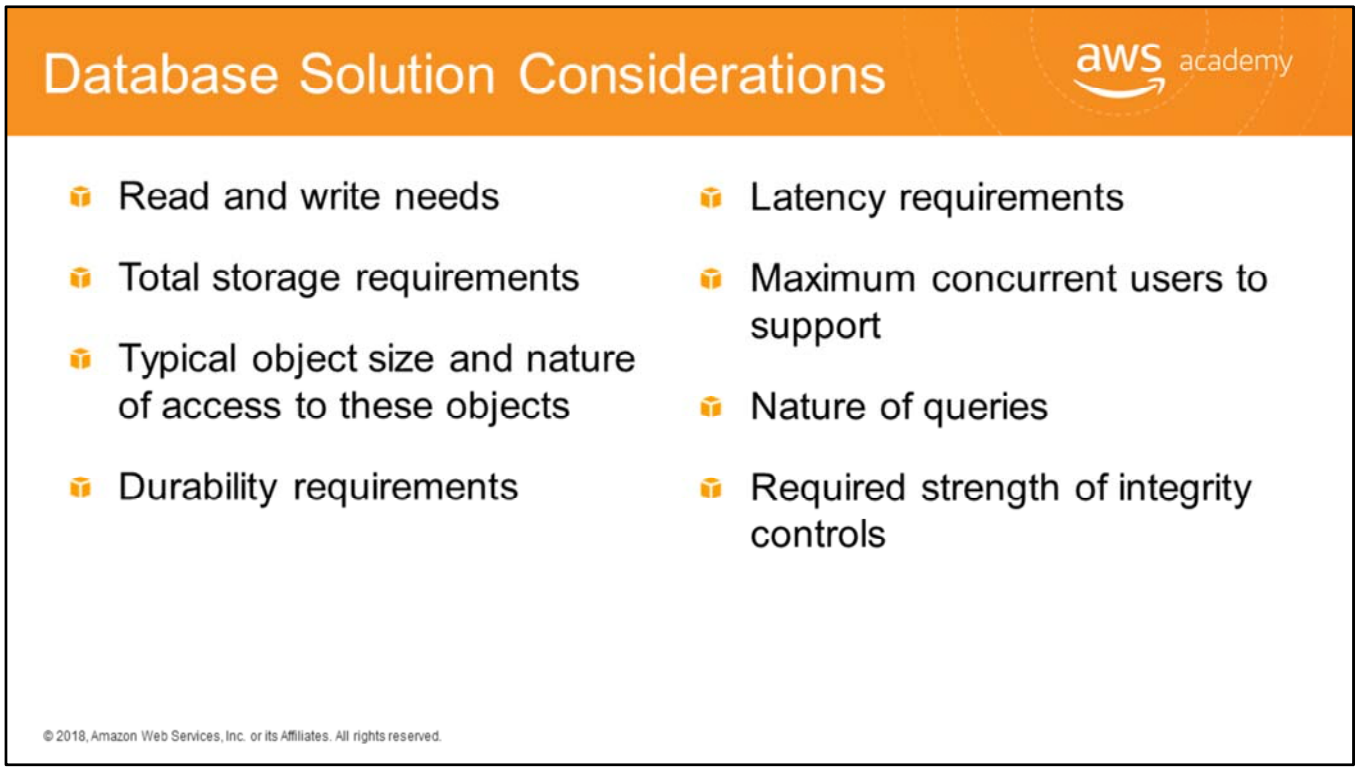
How should web accessible content be stored?
Let’s discuss using NoSQL databases such as Amazon DynamoDB.



It's important to choose the right database solution. If you already made a large investment in a structured query language—or SQL—database engine, it doesn't mean that it's the right option for every project. You should match the technology to your workload needs, not the other way around. There are some solutions where NoSQL would be preferred, such as storing session data or shopping cart data.

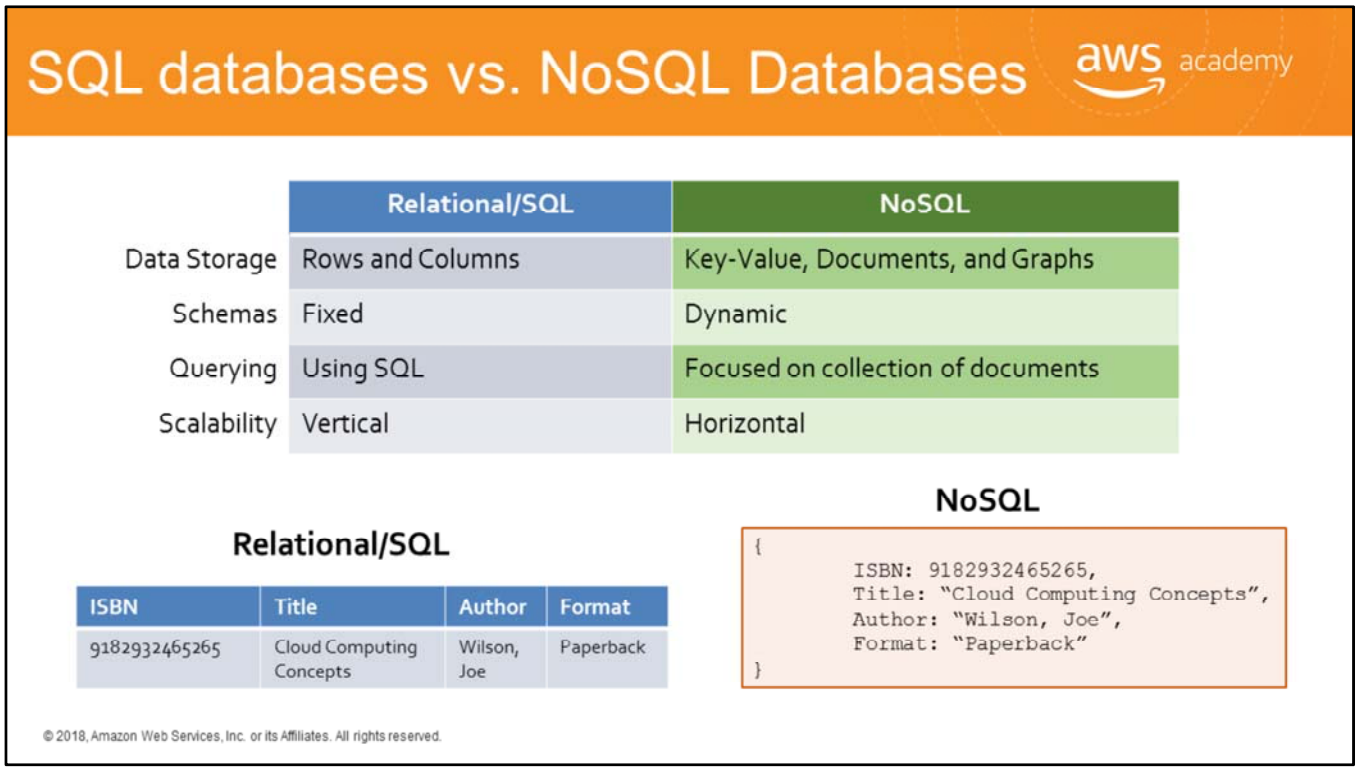
In traditional data centers and on-premises environments, limitations on what hardware and licenses are available can constrain your choice of data store solutions.

AWS offers data store options that provide more flexibility in how you choose your data store. You can choose from an array of relational database engines, NoSQL solutions, data warehousing options, and search-optimized data stores.



You should consider a number of things when you choose database solutions:

- Read and write needs
- Total storage requirements
- Typical object size and the nature of access to these objects
- Durability requirements
- Latency requirements
- Maximum concurrent users to support
- Nature of queries
- And the required strength of integrity controls



Let's break down the differences between a relational or SQL database and a NoSQL database. A SQL database stores data in rows and columns. Rows contain all the information about one entry, and columns are the attributes that separate the data points. A SQL database schema is fixed: columns must be locked before data entry. Schemas can be amended if the database is altered entirely and taken offline. Data in SQL databases is queried by using SQL, which stands for structured query language, which can allow for complex queries. SQL databases scale vertically by increasing hardware power.

A NoSQL database stores data by using one of many storage models, including key-value pairs, documents, and graphs. NoSQL schemas are dynamic. A row doesn't have to contain data for each column. Data in NoSQL databases is queried by focusing on collections of documents. NoSQL databases scale horizontally by increasing servers.

The slide shows an example of how data is organized in a SQL database versus a NoSQL database. It's the same data, but it's organized differently. If you have data that is long-term or permanent, SQL would be the better choice. For temporary data—such as website session data or a shopping cart— then NoSQL would be the better choice.

NoSQL Databases

- Can be an alternative to relational databases for some types of applications.
- Can process large amounts of data with high availability (depending on the NoSQL solution, configuration, and architecture).
- Form a broad category with different implementations and data models.
- Features distributed fault tolerance.

NoSQL databases:

- Can be an alternative to relational databases for some types of applications.
- Can process large amounts of data with high availability, depending on the NoSQL solution, and its configuration and architecture.
- Form a broad category with different implementations and data models.
- Features distributed fault tolerance.

NoSQL Databases

- Does your application need transaction support, ACID compliance, joins, and SQL?
- Can your application do without these for all, some, or part of its data model?
  - Refactor database hotspots to NoSQL solutions.
- Can offer increases in flexibility, availability, scalability, and performance.

Learn more.

Does your application need transaction support, ACID compliance, joins, or SQL?
Can your application do without these items for all, some, or part of its data model? Even though part of your data model might require SQL, it doesn't mean that all of the

data model will require SQL. You could divide your data model so that it uses the relational database for the data you need to query in a specific way or that you need to structure in a certain way. All the other data could be moved to a NoSQL database. You could also refactor database hotspots to NoSQL solutions.

NoSQL databases can offer increases in flexibility, availability, scalability, and performance.

For reference, ACID refers to atomicity, consistency, isolation, and durability, which are the four primary attributes that guarantee

that database transactions are processed reliably.

To learn more, go to the following page about the ACID model:
http://databases.about.com/od/specificproducts/a/acid.htm

NoSQL Databases

NoSQL on Amazon EC2:

- Cassandra, HBase, Redis, MongoDB, Couchbase, and Riak.

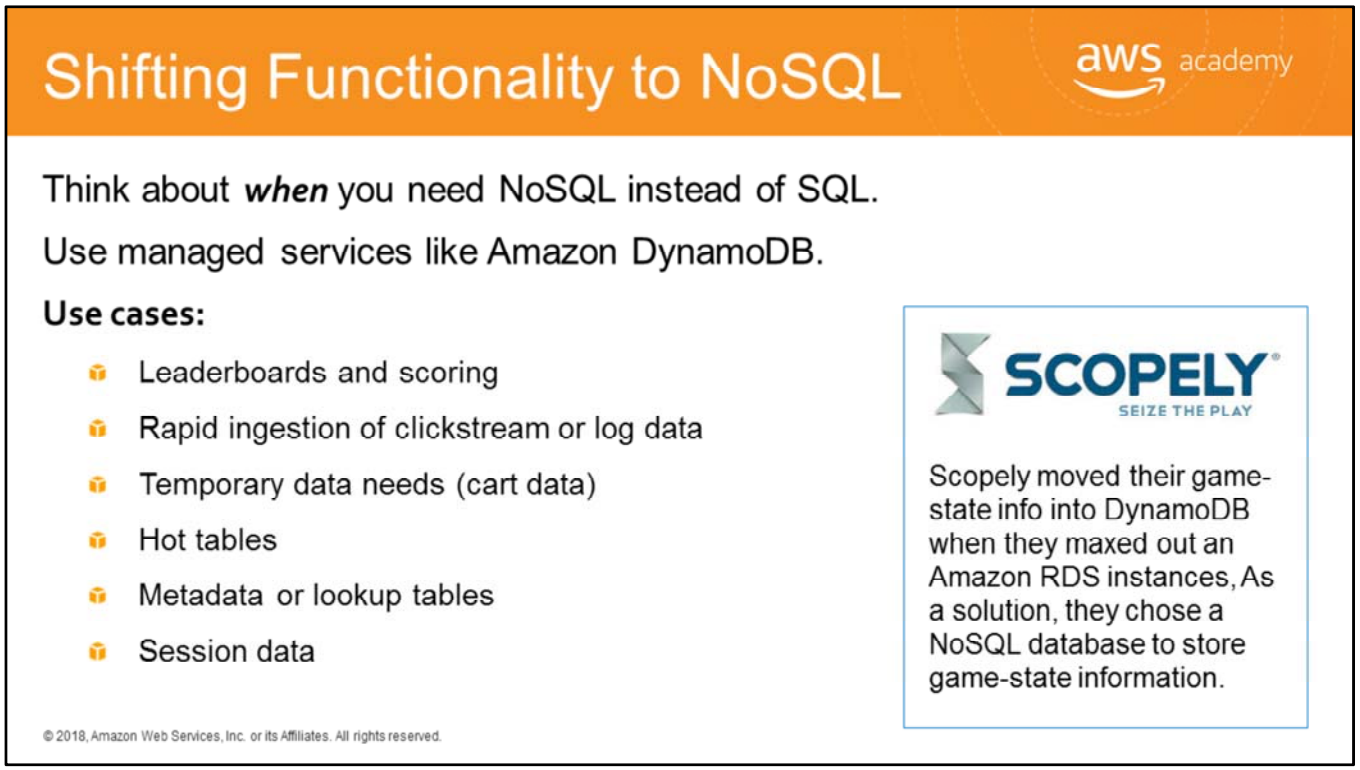
Managed NoSQL:

- Amazon DynamoDB
- Amazon Neptune
- ElastiCache with Redis
- Amazon EMR supports HBase

Let’s review some additional types of NoSQL databases, other than Amazon DynamoDB.

You can host your very own NoSQL database on an Amazon EC2 instance. NoSQL databases that you might consider include Cassandra, HBase, Redis, MongoDB, Couchbase, and Riak.

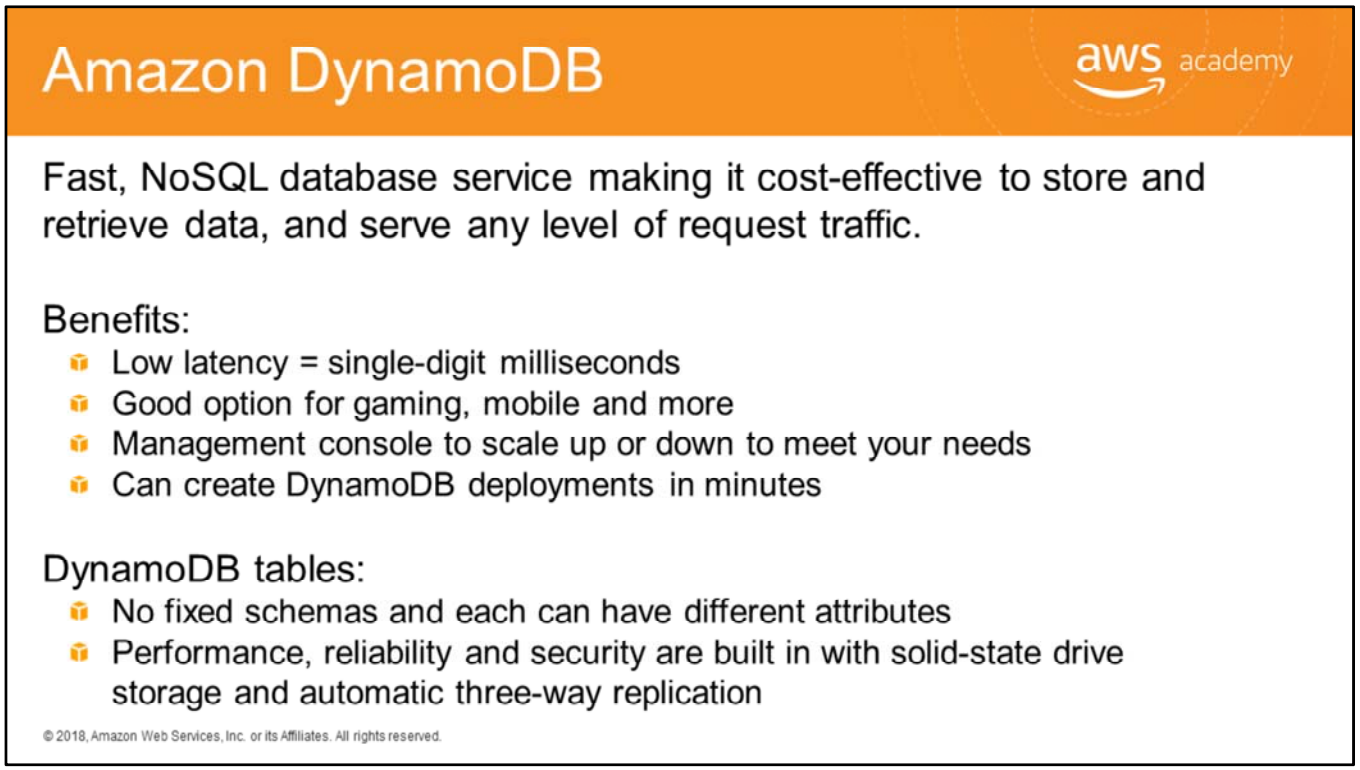
For managed NoSQL databases, we offer Amazon DynamoDB; Amazon Neptune, which can be used for graphs; Amazon ElastiCache supported by Redis; and Amazon EMR, which supports HBase.



What about shifting database functionality to a NoSQL database? When would you use NoSQL over SQL?

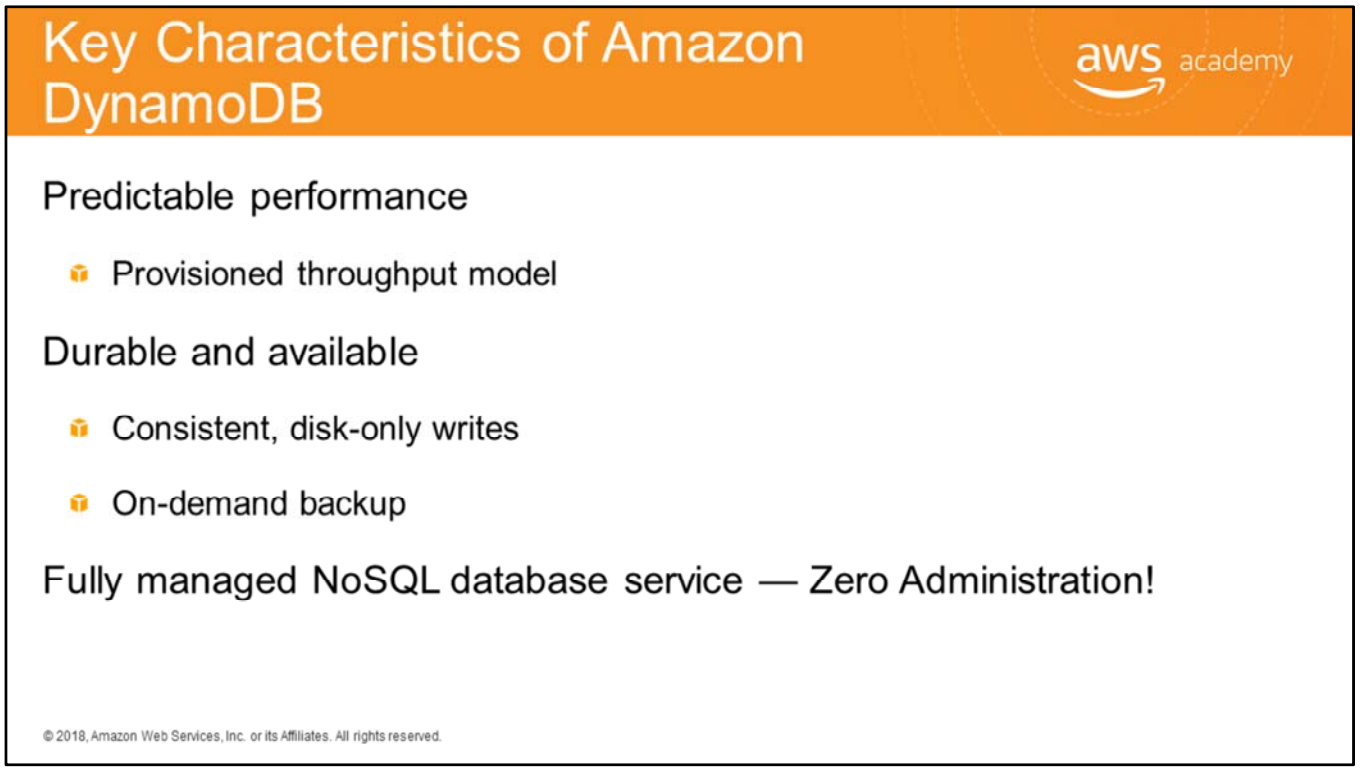
In addition to using managed services like Amazon DynamoDB, other options include:
- Leaderboards and scoring
- Rapid ingestion of clickstream or log data
- Temporary data needs, such as shopping cart data
- Hot tables
- Metadata or lookup tables
- And session data

As an example, an AWS customer, Scopely, decided to move game-state information into DynamoDB when they exceeded the capacity of an Amazon RDS instance. Instead of spending time sharding or optimizing Amazon RDS, they decided that a NoSQL database was what they needed to store game-state information.



Amazon DynamoDB has other key characteristics. It is a fast, NoSQL database service that makes it straightforward and cost-effective to store and retrieve any amount of data, and serve any level of request traffic. Its throughput and single-digit-millisecond low latency make it a good option for gaming, advertising technology, mobile, and many other applications. DynamoDB delivers seamless throughput and storage scaling via an API, and it has a management console that you can use to scale up or down to meet your needs. Many AWS customers have created DynamoDB deployments in a matter of minutes. These deployments were able to serve trillions of database requests per year.

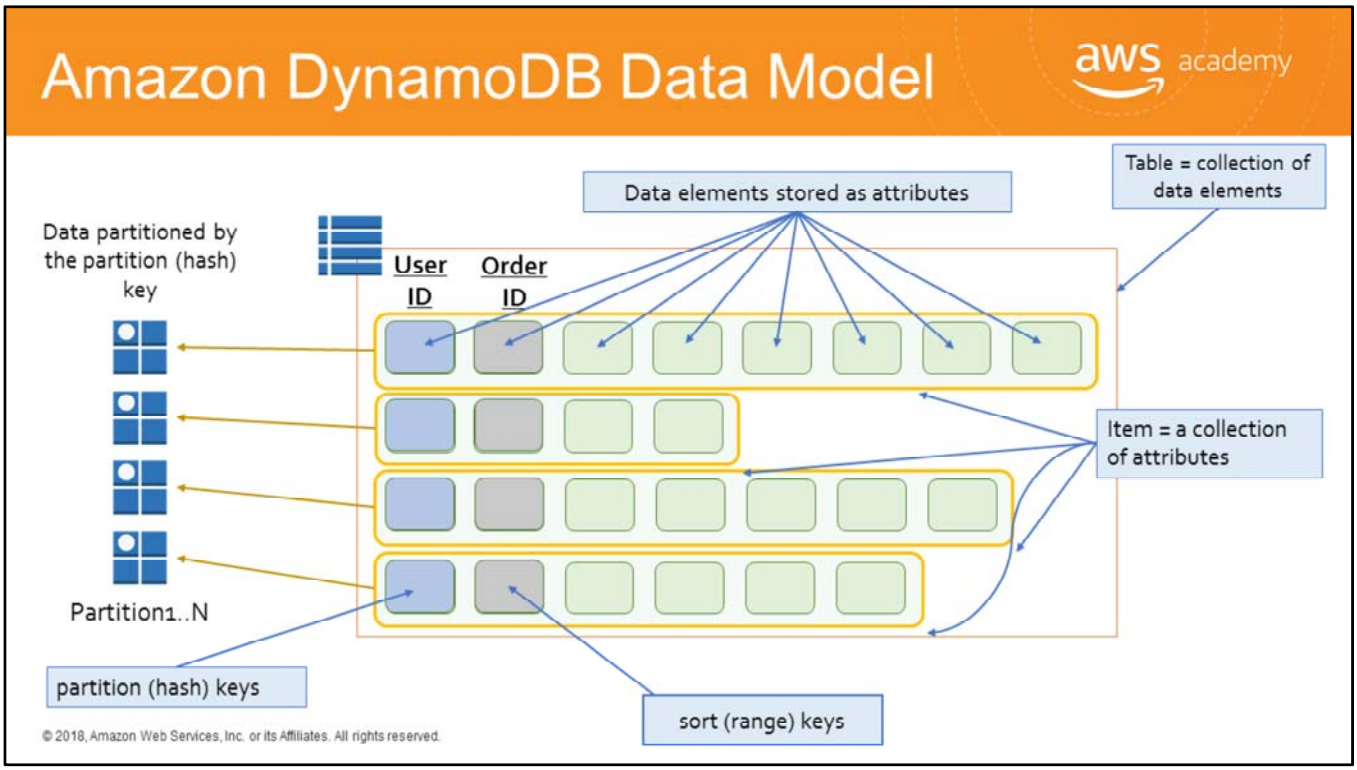
DynamoDB tables do not have fixed schemas, and each item can have a different number of attributes. Multiple data types add richness to the data model. Secondary indexes add flexibility to the queries that you can perform without affecting performance. Performance, reliability, and security are built in, with solid-state drive storage and automatic three-way replication. DynamoDB uses proven cryptographic methods to securely authenticate users and prevent unauthorized data access.



A few more characteristics of Amazon DynamoDB include its predictable performance with a provisioned throughput model.

It's also durable and available for consistent, disk-only writes and on-demand backups.

Amazon DynamoDB is a fully managed NoSQL database service with little administration.

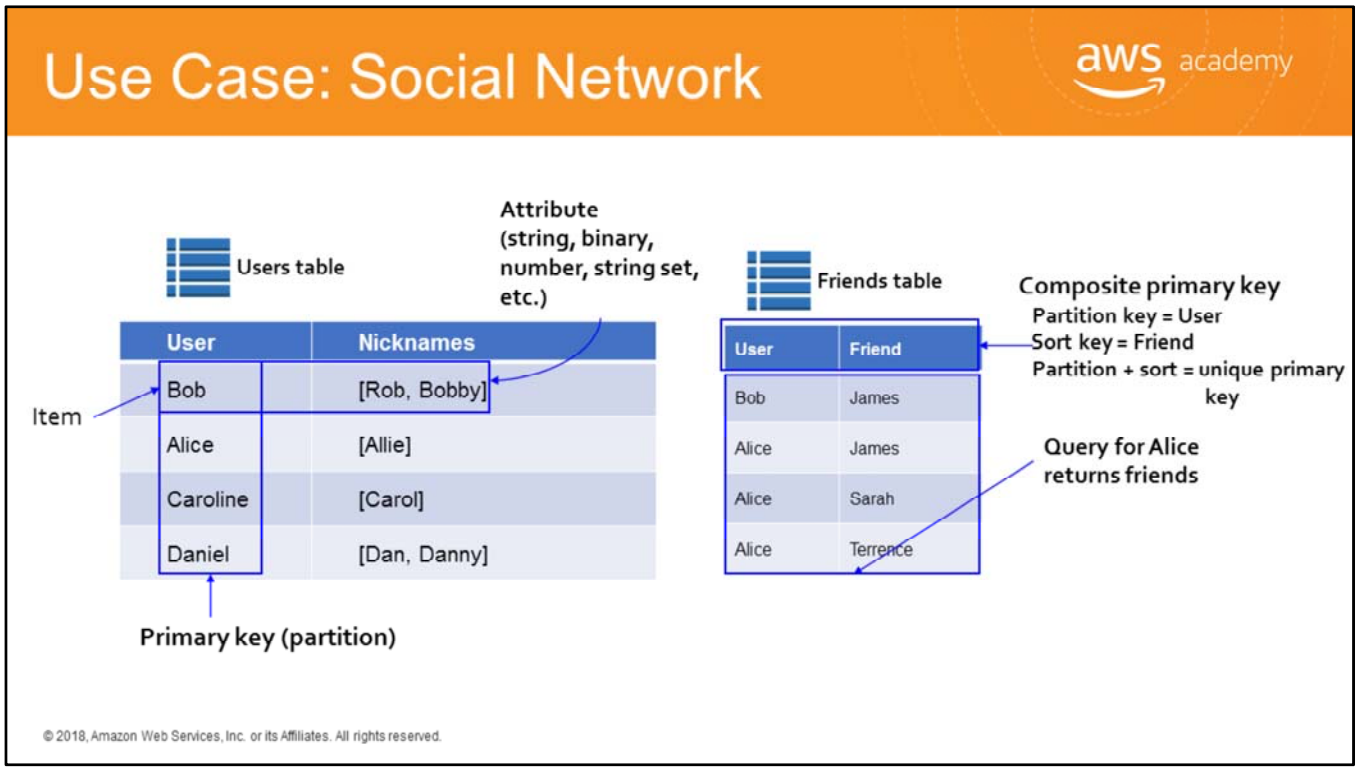


The world of DynamoDB does not have schemas. In a table, you have items. An item has variable attributes that have key-value pairs, which is similar to associative array or hash. In DynamoDB, they are called key and attribute.

DynamoDB supports key-value GET and PUT operations by using a user-defined primary key, which is a partition key, and is also known as a hash key. The primary key is the only required attribute for items in a table, and it uniquely identifies each item. You specify the primary key when you create a table. In addition, DynamoDB provides flexible querying by allowing queries on non-primary key attributes by using global secondary indexes and local secondary indexes.

A primary key can be a single-attribute partition key or a composite partition-sort key. A composite partition-sort key is indexed as a partition key element and as a sort—also known as range—key element. This multi-part key could be a combination of UserID, which is a partition, and Timestamp, which is a sort. By holding the partition key element constant, you can search across the sort key element to retrieve items. For example, you can retrieve all items for a single UserID across a range of timestamps.

Auto-partitioning occurs with data set size growth, and as the capacity that you provision increases.



Take a social network as an example. Users have friends, so you create tables for Users and Friends. In the Users table, you use User as a partition key to query the data. For each item of a user, you have attributes. In this example, attributes are nicknames. As you can note from the example, the attributes do not all have to be the same length. Attributes can be of type string, number, binary, string-set, number-set, etc.

In the Friends table, you use a composite primary key. The partition key is User and the sort key is Friend. If you combine the user partition key and the friend sort key, you get a unique primary key for identifying that friendship.

The item size cannot exceed 400 KB, which includes both the binary length of the attribute name—or UTF-8 length—and the binary length of the attribute value. The attribute name counts towards the size limit. For example, consider an item that has two attributes: one attribute that is named shirt-color with a value of R, and another attribute that is named shirt-size with a value of M. The total size of that item is 23 bytes.

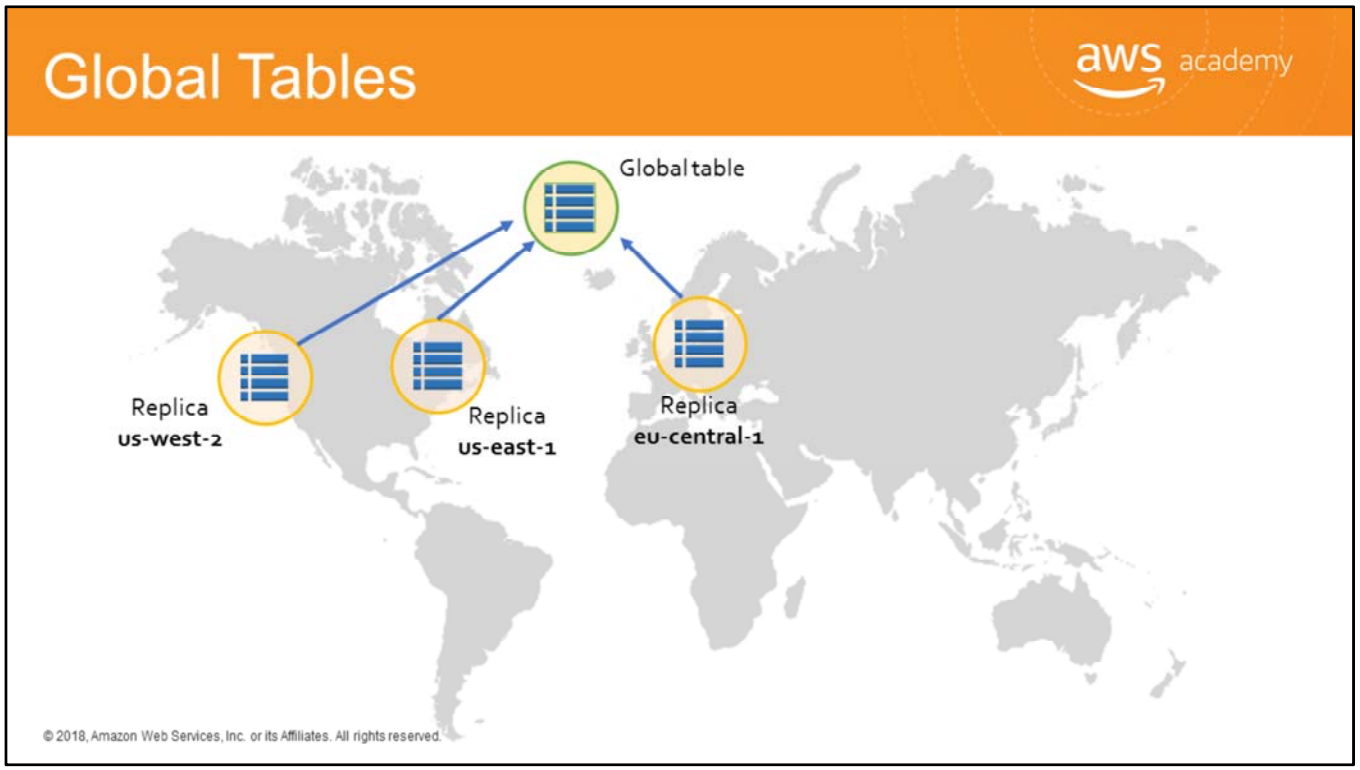
Amazon DynamoDB Consistency

aws academy

- DynamoDB synchronously replicates data across three facilities within an AWS Region.
- You can specify, at the time of the read request, whether a read should be **eventually** or **strongly** consistent.

Let’s discuss the consistency of Amazon DynamoDB. You can synchronously replicate data across three facilities within an AWS Region. Also, at the time of the read request, you can specify whether a read operation should be eventually or strongly consistent.

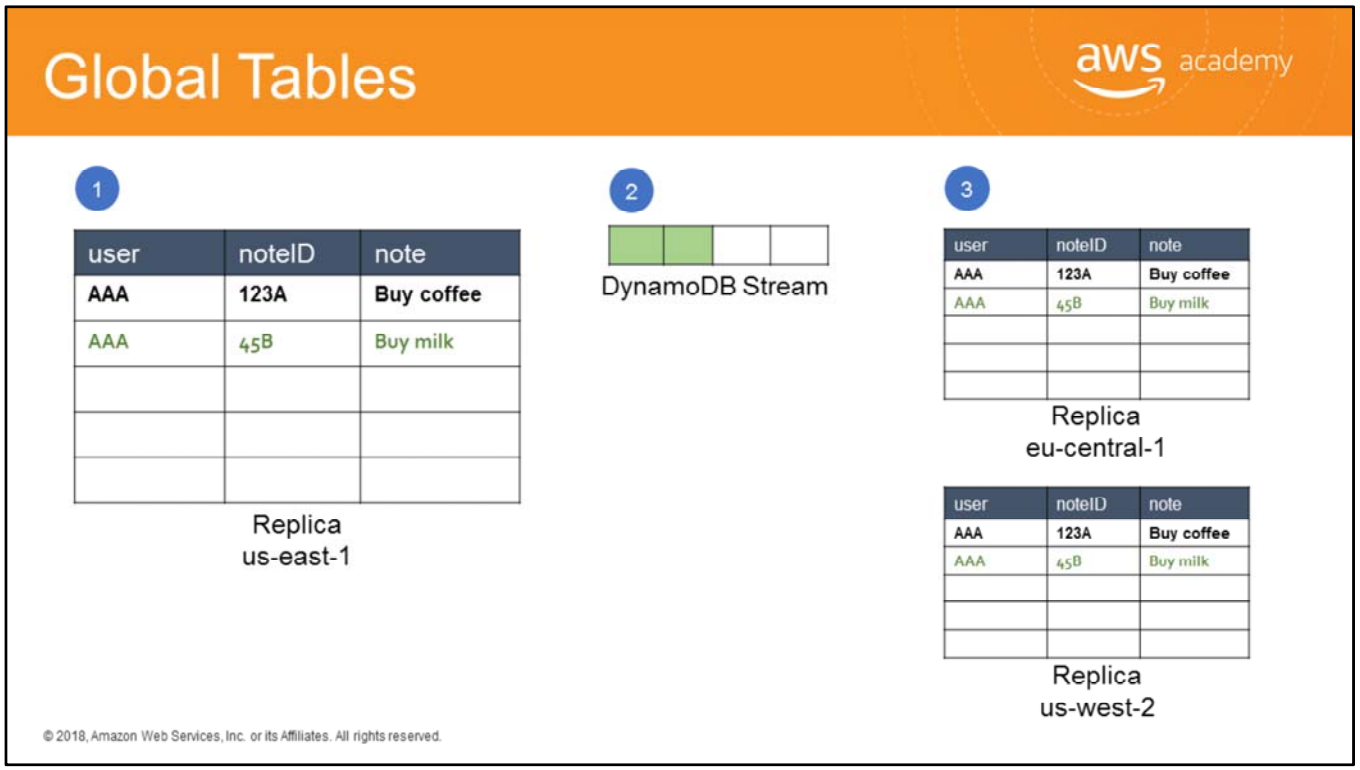
Keep in mind that if you do require strongly consistent read operation, it’s a 50 percent increase in cost.



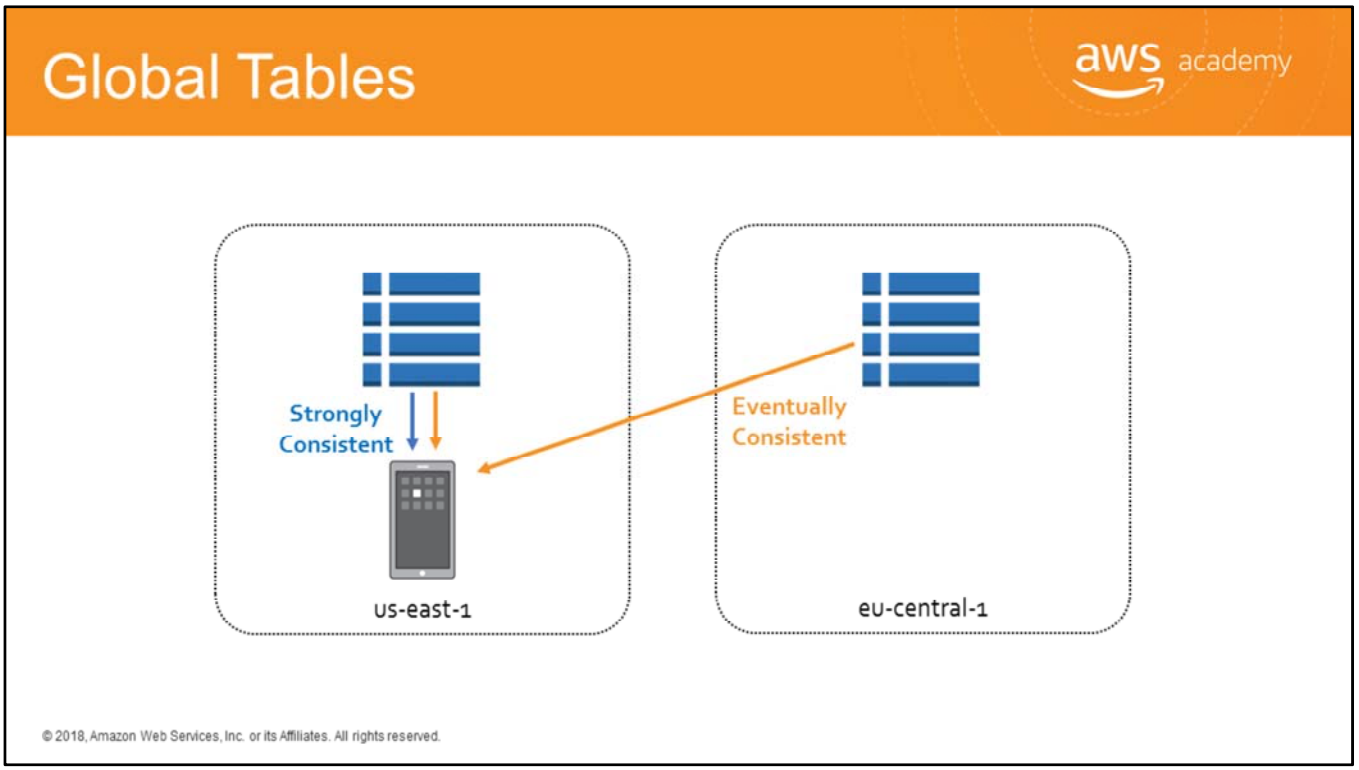
This slide shows an example of a global table that has replicas across three different Regions. A global table is a collection of one or more DynamoDB tables, which are all owned by a single AWS account. Each table in the global table is identified as a replica table.

A replica table—or replica, for short—is a single DynamoDB table that functions as a part of a global table. Each replica stores the same set of data items. Any given global table can have only one replica table per Region, and every replica has the same table name and the same primary key schema.

Amazon DynamoDB global tables provide a fully managed solution for deploying a multi-Region, multi-master database without having to build and maintain your own replication solution. When you create a global table, you specify the AWS Regions where you want the table to be available. DynamoDB performs all of the necessary tasks to create identical tables in these Regions, and propagate ongoing data changes to all of them.



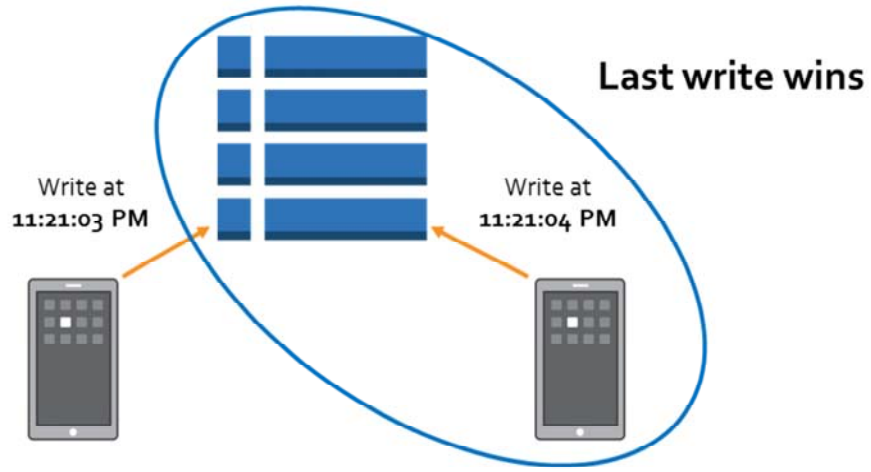
In the example on this slide, we have a replica in the us-east-1 Region, an Amazon DynamoDB Stream, and two additional replicas. Any changes that are made to any item in any replica table will be replicated to all of the other replicas within the same global table. In a global table, a newly written item, such as the one on this slide, is usually propagated to all replica tables within seconds. With a global table, each replica table stores the same set of data items. DynamoDB does not support partial replication of only some of the items.



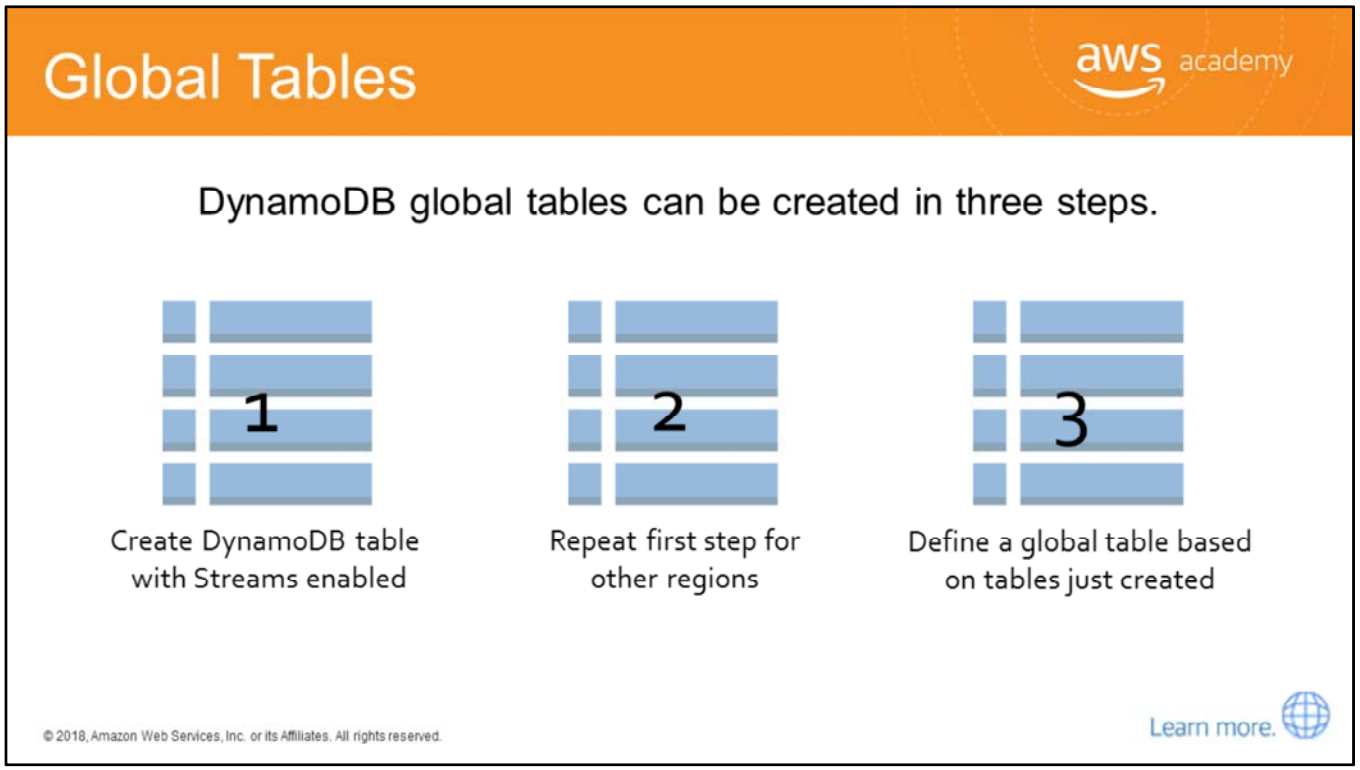
In the table that is identified as a global table—which is represented in the diagram as us-east-1--the writes and reads will always be strongly consistent. However, us-central-1, which is a replica, will only be eventually consistent for writes.

An application can read and write data to any replica table. If your application only uses eventually consistent reads, and only issues reads against one AWS Region, then it will work without any modification. However, if your application requires strongly consistent reads, then it must perform all of its strongly consistent reads and writes in the same Region. DynamoDB does not support strongly consistent reads across AWS Regions. Therefore, if you write to one Region and read from another Region, the read response might include stale data that doesn't reflect the results of recently completed writes in the other Region.

Global Tables



Conflicts can arise if applications update the same item in different Regions at about the same time. In this example, one user writes at 11:21:03 PM, and a second user writes to that exact same item at 11:21:04 PM. To ensure eventual consistency and settle any conflict, DynamoDB global tables use a last write wins reconciliation between concurrent updates, where DynamoDB makes a best effort to determine the last writer. With this conflict resolution mechanism, all of the replicas will agree on the latest update, and converge toward a state in which they all have identical data.

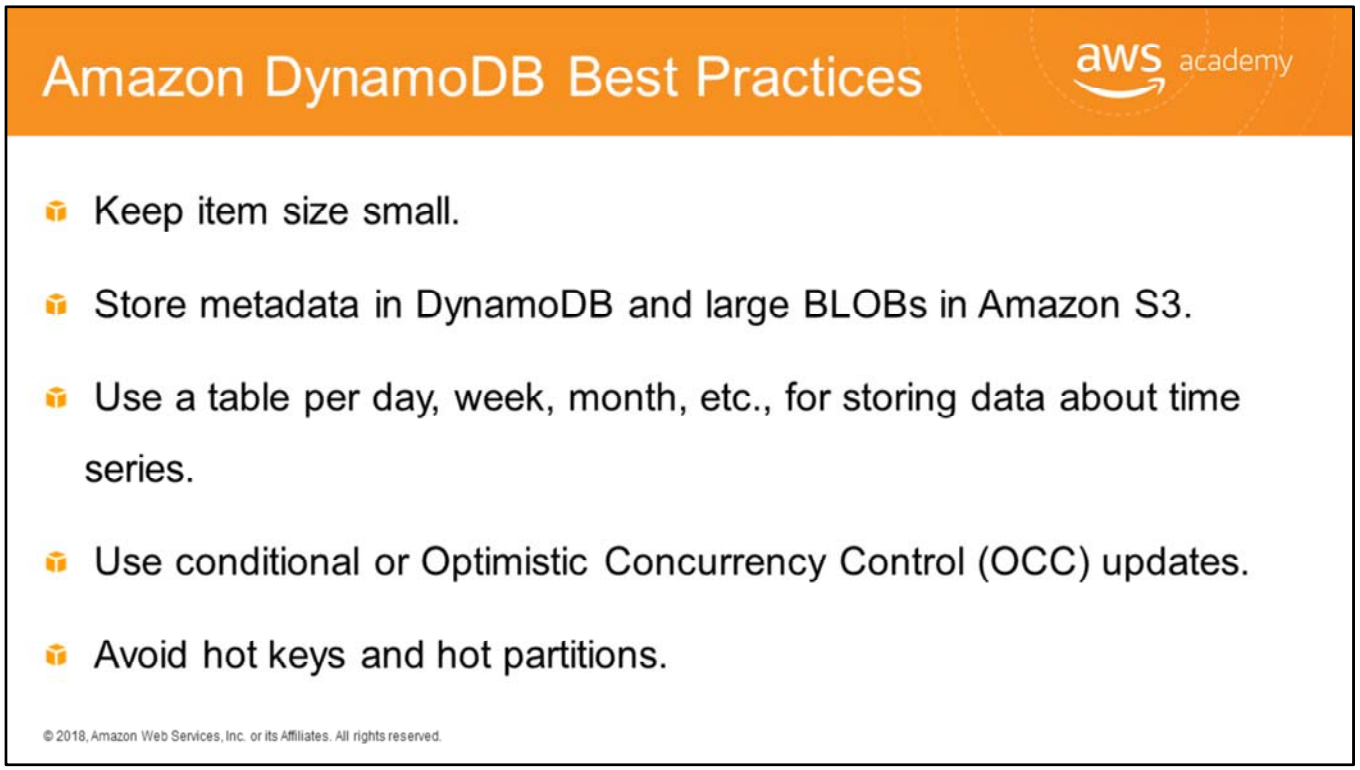


How do we enable global tables within Amazon DynamoDB?

DynamoDB global tables can be created in three steps:

• The first step is to create an ordinary DynamoDB table —with DynamoDB Streams enabled—in an AWS Region.
• Next, repeat the first step for every other AWS Region where you want to replicate your data, and create other DynamoDB tables.
• Finally, based upon the tables that you have created, you can define a DynamoDB global table.

The AWS Management Console automates these tasks, so you can create a global table quickly.
For detailed instructions, go to the following page.
http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/globaltables.tutorial.html



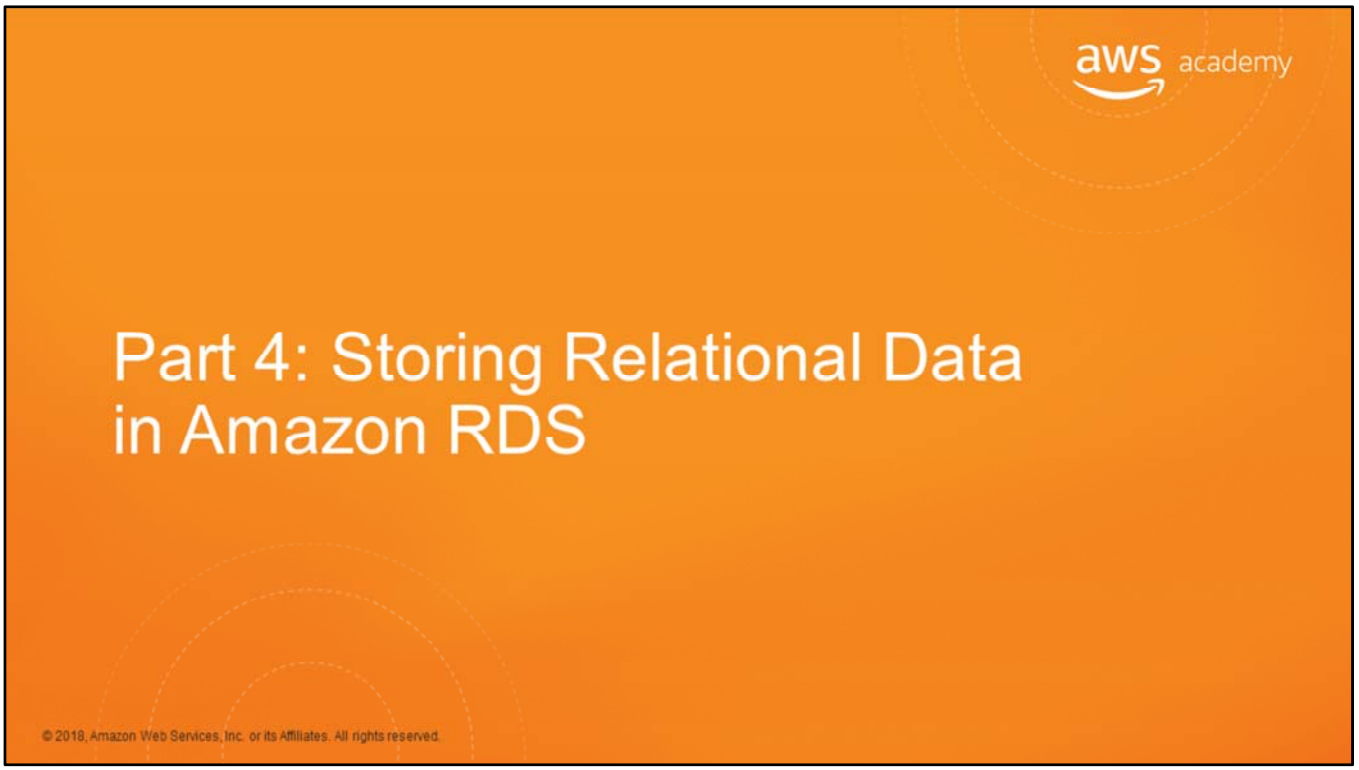
Some best practices to remember regarding Amazon DynamoDB include:

- Keeping the item size small.
- Store metadata in DynamoDB and large binary large objects—or BLOBs—in Amazon S3.
- Use a table per day, week, month, etc., for storing data about time series.
- Use conditional or optimistic concurrency control updates.
- And avoid hot keys and hot partitions.

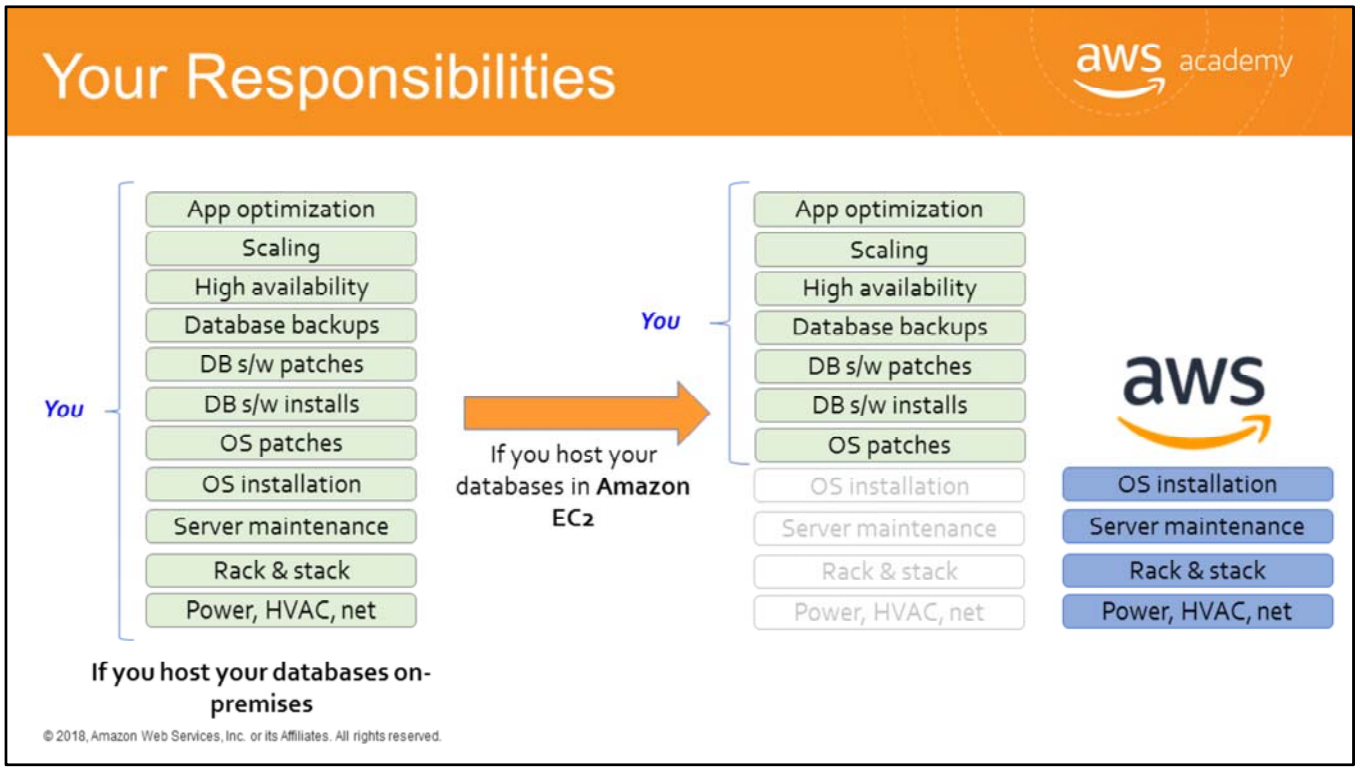
Optimistic concurrency control—or OCC—is a concurrency control method that is applied to transactional systems, such as relational database management systems and software transactional memory. OCC assumes that multiple transactions can frequently be completed without interfering with each other. While they are running, transactions use data resources without acquiring locks on those resources. Before each transaction is committed, the transaction verifies that no other transaction has modified the data it has read. If the check reveals conflicting modifications, the transaction that is being committed rolls back and can be restarted.

OCC is generally used in environments with low data contention. When conflicts are rare, transactions can be completed without the expense of managing locks, and without having transactions wait for other transaction locks to clear. This method leads to higher throughput than other concurrency control methods. However, if contention

for data resources is frequent, the cost of repeatedly restarting transactions hurts performance significantly. It is commonly thought that other concurrency control methods have better performance under these conditions. However, locking-based—or pessimistic—methods also can deliver poor performance because locking can drastically limit effective concurrency even when deadlocks are avoided.



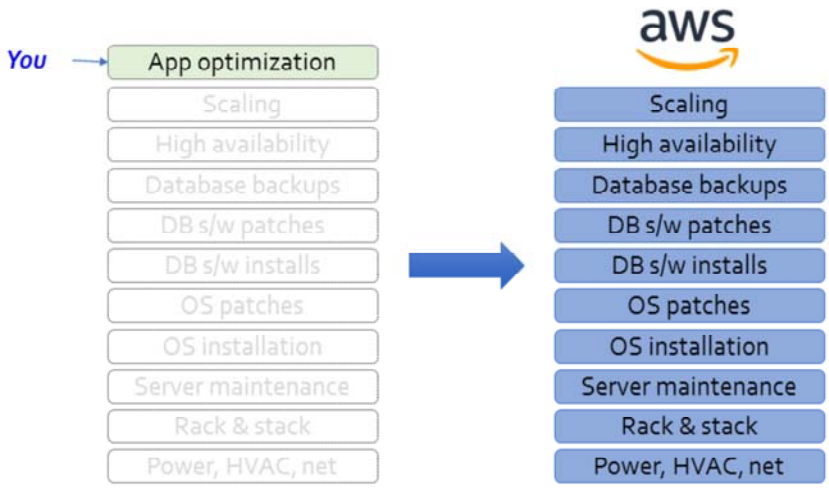
Let’s discuss storing relational data in Amazon RDS.



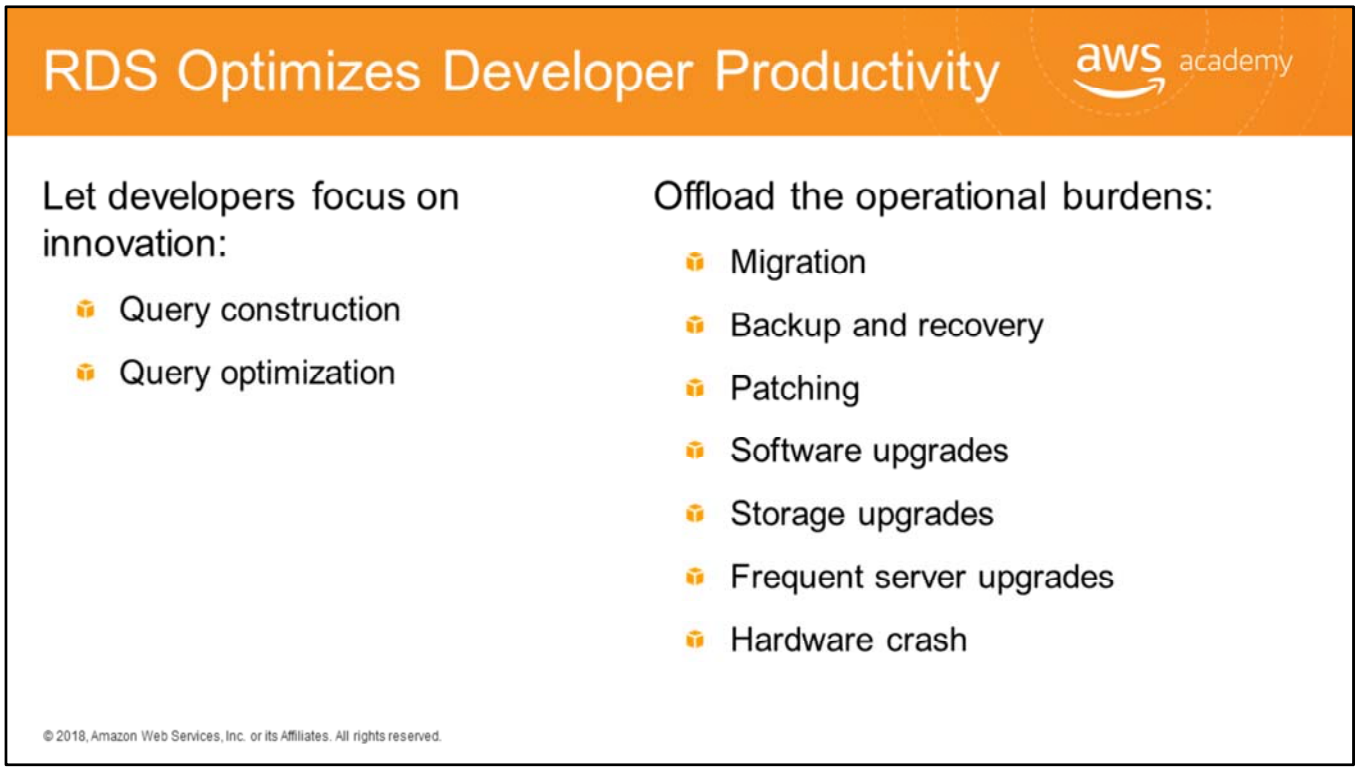
If you host your own database on-premises, you must take care of everything. Your responsibilities include optimizing your application, scaling, ensuring high availability, installing operating system—or OS—software and maintenance, etc.

If you host your databases in Amazon EC2, you don't need to install the OS, maintain the server, rack and stack servers, maintain the power supply, etc.

Choosing Amazon RDS



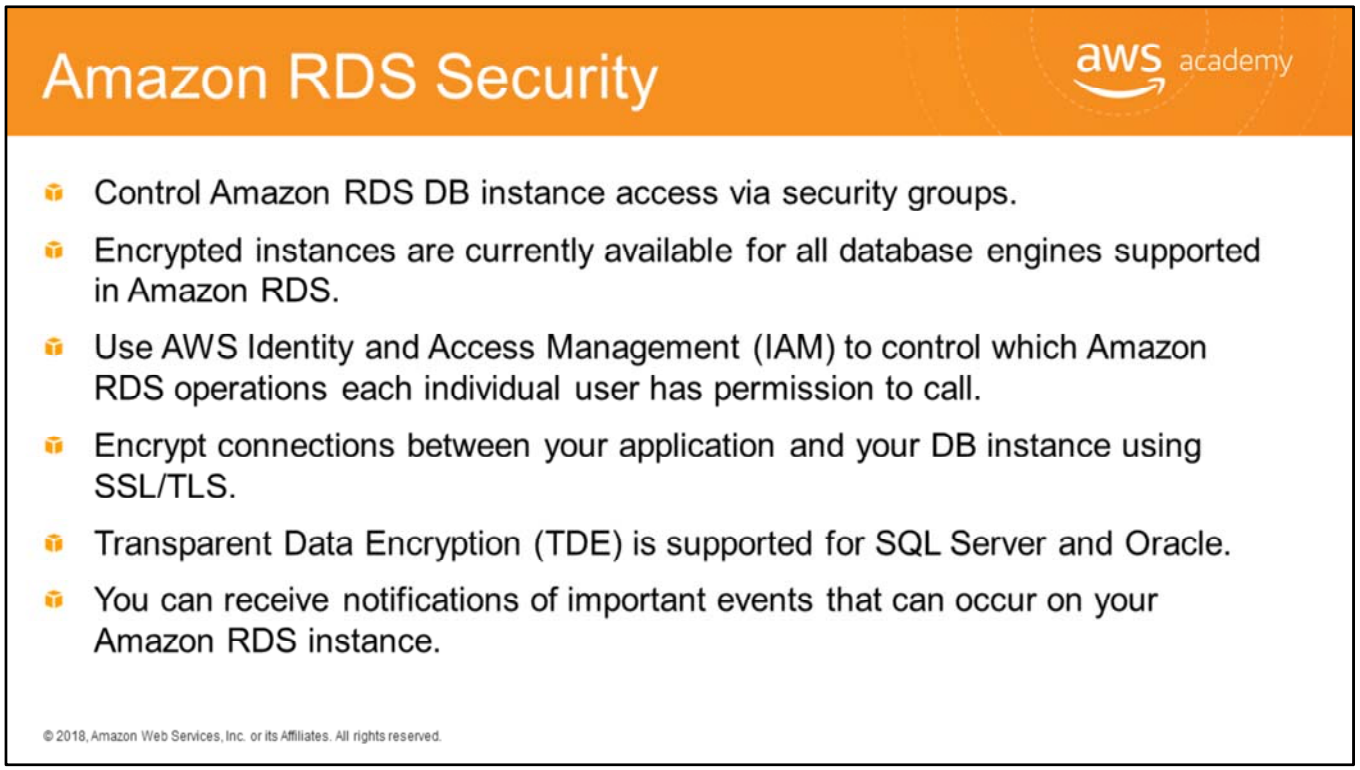
If you choose a managed DB service like Amazon RDS, your only concern is optimizing your application. We take care of the rest, so the time that you gain by offloading undifferentiated labor to us can be used to optimize your application.



Amazon RDS is a good option for developers because it allows developers to focus on innovation, such as query construction and optimization.

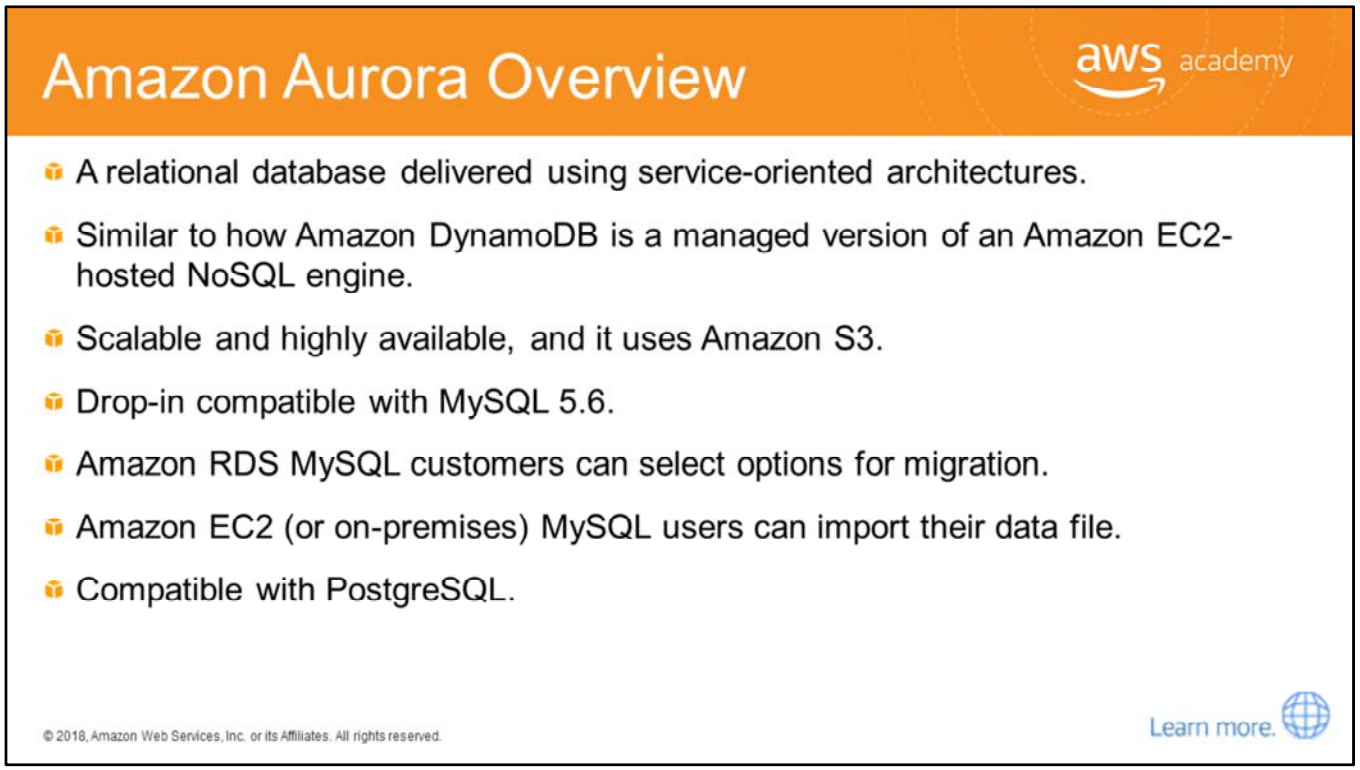
Using Amazon RDS lets you offload the following operational burdens responsibilities:

- Migration
- Backup and recovery
- Patches
- Software upgrades
- Storage upgrades
- Frequent server upgrades
- And hardware crashes



What about security for Amazon RDS?

- You can control an Amazon RDS database instance by enabling security for Amazon RDS.
- If you enable security, the backups or snapshots of your data that you create will automatically be encrypted.
- Encrypted instances are currently available for all database engines that are supported in Amazon RDS.
- Use AWS Identity and Access Management—or IAM—to control which Amazon RDS operations each individual user can call.
- You can encrypt connections between your application and your DB instance by using SSL/TLS.
- Transparent Data Encryption—or TDE—is supported for SQL Server and Oracle.
- You can receive notifications of important events that occur on your Amazon RDS instance.



Amazon Aurora is a relational database that is delivered by using service-oriented architectures.

- It is similar to how Amazon DynamoDB is a managed version of an Amazon EC2-hosted NoSQL engine.
- It is scalable and highly available, and it uses Amazon S3.

- It has drop-in compatibility with MySQL v5.6.
- Existing applications “just work.”
- Amazon RDS customers that use MySQL can select options for migration.
- Amazon EC2 or on-premises MySQL users can import their data file.
- And it is compatible with PostgreSQL.

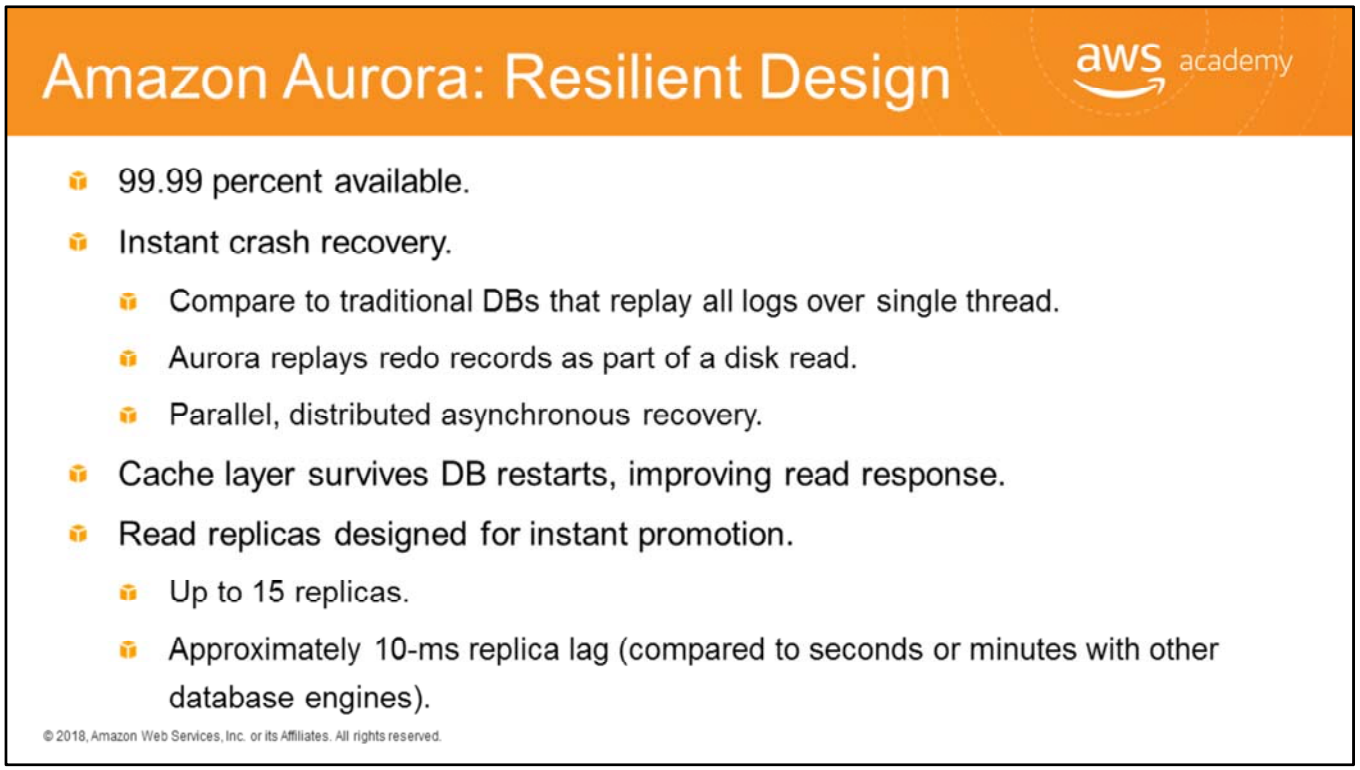
To learn more, go to the following Amazon Aurora page.
https://aws.amazon.com/blogs/aws/highly-scalable-mysql-compat-rds-db-engine/

Amazon Aurora: Scalable Performance

aws academy

- Is integrated with Amazon S3 for continuous backup.
  - Six copies replicated across three Availability Zones.
- Moves logging and storage layer into multi-tenant, scalable service layer.
- Uses SSD data plane up to 64 TB max DB volume
  - Pay only for consumed amount.
  - Added in 10GB increments.

Amazon Aurora is integrated with Amazon S3 for continuous backup. Six copies are replicated across three Availability Zones.
You can move your logging and storage layer into a multi-tenant, scalable service layer. Amazon Aurora is also powered by solid state drive data planes of up to 64 TB per volume. You can add more storage in 10 GB increments in an on-demand basis. You only pay for the amount that you consume.



Amazon Aurora was built with a resilient design, and it offers an availability of 99.99 percent. It also has instant crash recovery.

- Aurora can be compared to traditional databases that replay all logs over a single thread.
- Aurora replays redo records as part of a disk read.
- And Aurora includes parallel, distributed asynchronous recovery.

The cache layer survives database restarts, which improves the read response.

Read replicas are designed for instant promotion.

- Promotion can occur with up to 15 replicas.
- Aurora has a 10-millisecond replica lag, which can be compared to a lag of seconds or minutes that can occur with other database engines.